NANYANG TECHNOLOGICAL UNIVERSITY

HIGH-PERFORMANCE COMPUTER ANIMATION RENDERING FRAMEWORK

CHONG SOON KEONG ANTHONY

SCHOOL OF COMPUTER ENGINEERING

2010

HIGH-PERFORMANCE COMPUTER ANIMATION RENDERING FRAMEWORK

CHONG SOON KEONG ANTHONY

2010

HIGH-PERFORMANCE COMPUTER ANIMATION RENDERING FRAMEWORK

CHONG SOON KEONG ANTHONY

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Master of Engineering (Computer Engineering)

2010

A B S T R A C T

Rendering computer animation frames is a very time consuming job. Using parallel computing on clusters and so-called render farms is a common solution to this problem. However, there also exists a significant obstacle in sending large gigabytes computer animation files across the Internet.

In this report, we describe how Grid computing can be used for computer animation rendering. We propose a framework for Grid rendering services, describe its implementation, and present the results and statistics. We also propose an efficient lossless compression algorithm devised for transferring gigabytes of 3D scene representation files in *RenderMan*® and *mental images*® data formats. We illustrate how the algorithm works within a developed framework for Grid rendering services, and present the results and statistics. This compression algorithm has been filed for patent in Singapore.

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude to my advisor, Associate Professor Alexei Sourin, for his support, contributions, guidance and most important of all giving me the opportunity to pursue my research studies in the School of Computer Engineering at NTU. His trust in me and enthusiasm has always been a great motivation and a driving force behind this project.

I would also like to express my gratitude to Dr. Konstantin Levinksi for many inspiring discussions about computer graphics and computer network all these years. He has helped me a lot in this research.

I would also like to thanks GSCT at KAIST, Korea where I am encouraged to further develop my research interest and have the opportunity to work along with talented young professors in GSCT. They have made GSCT a very sociable and friendly atmosphere to work in.

Finally, I would like to thank my wife, Maggie for her unreserved love, support trust and patience.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION AND APPROACH

The motivation behind this research is to develop an inexpensive, user-friendly web-based rendering system where computationally expensive computer animation rendering work can be performed with ease and inexpensively by animators from small to middle scale 3D animation studios. There are 11 stages in computer animation production pipelines.

1) Story creation. This is the most important part of any animation. A superior visual effects movie will never attract viewers without a good storyline.

2) Visual Development. This is a brainstorming stage to design how the scenes will look like including creation of character (superheroes, evil characters, bugs, etc), environments (grassland, desserts, sea, land, etc), props and so on for the movie scenes. This stage will often involve painters, sculptors, designers, illustrators, etc. with lots of drawings pasted up the wall.

3) Character design. This stage consists mostly of drawings or sculptures where the poses of body, facial expressions, key features from multiple points of view and the body movement of different characters are considered.

4) Storyboarding. This is the stage where the movie will be sketched on papers pasted on a wall/board. This stage often brings out the key scenes, specific how the camera moves and specific the character unrefined motion. Normally, this stage is to verify whether the storyline is clear and easily understood with characters being clearly portrayed.

5) Scene layout. This is the stage where the scenes will be designed with understanding of the camera pan and movement of the characters. This stage also involves creation of colours, textures and props for the scenes.

6) Modeling. Geometric models of the environment, characters and props created using computer modeling tools (examples, 3D Studio Max, 3D Maya, etc). Manual rigging is often done to specify its internal skeletal structure and to determine how the input motion (moving the skeleton's joints) deforms its surface. This process is to prepare the character for the animation stage.

7) Animation. This stage involves Keyframing- a technique to allow the animator to position the models at various points in the frame. The computer will then interpolate the remaining frames between these various points to form the animation sequences.

8) Shading and Texturing. Shading is the process of applying mathematical-created textures to the models or scenes. This can be done using specific tools like the Pixar's RenderMan Shading Language (RSL). For example, special required textures (dinosaur's skin, floor marble with unique design) or textures used for special effects (fire, smoke, dusts, etc) not easily available can be created using RSL. Texturing is the process of apply an image to the outer surface of the model.

9) Lighting. There are 4 common light sources to illuminate surfaces in animation: Point light, spot light, directional light and ambient light. Animators often rely on their experience, select the appropriate light sources, set the properties of each light source and position them accordingly to simulate the overall visual effects desired.

10) Rendering. Rendering is the process of generating an image from a 3D model. The rendered images will be blended seamlessly together to form an animation.

11) Post-Production. This final stage includes editing individual shots or scenes, created transitions, compositing and adding in special effects.

Based on understanding on the flow of the production pipelines, creating an animation can be a long process of hard work that involves large resources of manpower and cash flow. Computer animation rendering is a very computationally expensive task depending on the complexity of the modeling scenes which could take from a few minutes to a few hours for a single frame. One second of animation requires 30 sequential frames of images and a short advertisement of 30 seconds requires (30 x 30) 900 frames where a highly resolute single frame, for example, could take 1 hour of rendering time on a single computer. Therefore, in total it requires 900 hours (37.5 days) of rendering time. Animation studios have tried to distribute the rendering loads to a few more computers available in the studios. Based on industrial visits, we understand that most small studios have only less than 10 computers and most of them are highly utilized by other animators in the daytime. The rendering normally take place after office hours in the nights when the computers are available. Most small, middle scale studios cannot afford to accept multiple projects at the same time as their computing resources are limited therefore affecting their revenues. For big animation production, getting a movie onto the cinema screen is the only way to recoup all investment and to generate profits. Just imagine, assuming a movie frame took 6 hours on average for rendering by Pixar Animation Studio. 100 minutes (6000 seconds) animation movies at 30 frames per second, it will take an average of (6000 seconds*30 frames*6hours = 1080000 hours) 123 years for the movie to be produced. Without the render farm, no high quality movie can be seen in cinema in this century.

In this research work, we will propose a method how to use grid computing on general purpose computers for solving tasks of computer animation on demand. This approach is planned as inexpensive solution based on open source software from which many small and middle animation studios can benefit. The idea of the project is to develop two solutions 1) A portal where animators are about to upload their finished 3D models files from common 3D modeling software such as 3D Maya, Cinema4D and Mental Images for rendering via the internet. 2) Direct rendering from the 3D modeling software where a plug-in could be developed for the animators to submit their frames while still doing their animation jobs. The second solution is for animators to test-render part of their frames before submit the entire large frames to the system.

1.2 ORGANIZATION OF THESIS

This thesis is organized as follows. Chapter 2 introduces the background of the common scientific applications that have benefited from using grid computing technology and introduces one commercial available remote access render farm for the rendering of animation. Chapter 3 introduces our proposed grid-based rendering framework. Chapter 4 highlights the existing problem of having large generated datasets that will have problems uploading to the grid for rendering. This chapter introduces our novice data compression technology that are specially invented for uploading large datasets used in our grid-based rendering system. Chapter 5 demonstrated the final project implementation with the final results shown. Finally, chapter 6 summarizes and concludes this thesis.

CHAPTER 2

GRID COMPUTING TECHNOLOGY

2.1 INTRODUCTION

In previous chapter, we proposed on developing a method how to use Grid computing on general purpose computers for solving tasks of computer animation on demand. However, in this chapter, we will like to introduce Grid computing technology that has also been employed in various application domains including computer animation rendering.

2.2 GRID COMPUTING AND ITS APPLICATION DOMAINS

Computational Grid (or just Grid) can be defined as an infrastructure with interconnected nodes of resources. A node is an access point of communication or computation. The resources could be databases, application servers, networks and storage devices. The function of the Grid is to enable sharing of geographically distributed heterogeneous resources between interconnected nodes. Many researchers and institutions involving in various disciplinary fields of research that requires high-performance computing have understood the benefits of using computational grid to assist in their tasks. Here, we also introduce some scientific applications in the area of simulation and visualization along with computer animation rendering where grid computing technology has been employed. Each of these applications demonstrated the importance and

usefulness of having distributed computing resources that would otherwise be difficult for the researchers to efficiently conduct the experiments and obtain the results in reasonable time.

1. RENDERING OF MOVIE FRAMES

High quality frames rendering for movie animations requires intensive computation and therefore involves a large amount of time. A high performance distributed rendering system [1] was presented to compute frame sequences on demand. Open-source POV-Ray ray tracer application is used to render animation frames and Globus toolkit has been chosen as grid architecture. A web portal was built to submit job to the grid. The files were uploaded on the remote server using a web client, and, when the computation was finished, the movie was downloaded from the remote server. A secure protocol (https) was used for the communication.

The POV-Ray input files were large, therefore GASS (Global Access to secondary storage) was implemented to move POV-Ray files among grid components. The resulting rendering took only 16 minutes with having 12 x Athlon + 2 x P3 computers in the grid for the rendering compared to the original time of 3 hours 22 minutes from a single Athlon computer.

The Department of Informatics at University of Sussex, for example, has just installed 150 new Core2 Duo machines across 3 laboratories [2]. To achieve this goal, the open source Condor High Throughput Computing software was selected and implemented as a desktop Grid computing solution.

For experiment, a 3D animated virtual tour of Fisher bourn Roman Villa using Blender modeling software was created. This animation took 7 hours to render on a single core machine. The same animation was rendering in 55 minutes on the render farm.

The Centre for Parallel Computing (CPC) [3] at the University of Westminster, London (http://rendering-portal.cpc.wmin.ac.uk) is test-driving their 250-CPU Grid. Blender modeling software is being used for the grid and they are inviting users to render their work there. However, the files are not protected from the public once submitted.

The introduction of render farm in a distributed rendering environment in Purdue University [4] allows a course dealing with animation or scientific visualization to reduce its bottleneck

associated with the rendering process. The purpose is to enable students to produce high quality animation assignments that were previously been restricted by the lack of rendering resources.

2. MEDICAL FIELD

High resolution is especially beneficial when visualizing detailed medical data. The ability to adequately view and interact with these complex datasets can provide a wealth of information. Deep View visualization system [5] used underlying software, Chromium [6] and OpenDx-MPI (distributed-parallel visualization system based on OpenDX) to perform computation under a Linux cluster to produce 3D geometry, renders the geometry to produce 2D pixels, and then transfer the pixels to be displayed on a video wall. OpenDx-MPI makes use of the system's high-performance intercommunications to transfer the data in parallel to dedicated visualization nodes within the same distributed system. Interactive and animated high resolution (3800 x 2000 pixels) digital images from the National Library of Medicine's Visible Male dataset were visualized by Deep View system.

3. EARTHQUAKE SIMULATION

A computer simulation project in [7] was developed to predict the ground motion for large basins during strong earthquake in the Greater Los Angeles and Kobe, Japan basins. The Quake ground motion simulation needs to run remotely on supercomputer because of the large massive dataset that was generated. As mentioned in the paper, a simulation of 60 seconds of earthquake can generate 6 TB of data. Visualization is needed to interpret this massive dataset. However, manipulating huge datasets interactively is difficult because local machine resources are limited.

The crucial implications are that large scientific datasets must reside on the remote site where they are computed and interactive visualization applications that manipulate these remote datasets must be distributed, with the computation partitioned across hosts at the remote and local sites.

In earthquake simulation visualization, each frame records the displacement amplitude (amount of motion at that spot in the earth) at each node of a large unstructured mesh. User at the local site requests data from some region of interest (ROI) in the dataset. The region of interest was

expressed both in space within a frame or in time across multiple frames where the result from a single frame is an image and result of multiple frames is an animation.

A distributed visualization framework [8] is based on the central notion of an active frame, which is an application-level mobile object that contains both application data and a program that manipulate the data. Active frame are processed by active frame servers running on hosts and local sites. Transformations are applied to the active frame arriving at the server where a new output active frame is generated. The address of the next active frame server is generated and the server sends the output frame to the next server in the grid.

4. FLUID FLOW SIMULATION

Large fluid dynamical simulations [9] are performed on systems with 62 Dell PCS, each with an Nvidia GeForce GPU and 400GB of storage, University of Minnesota's Academic and Distributed Computing Services (ACDS) managed to render a movie of 3200 frames from 1.4 TB dataset at full 13 MegaPixel PowerWall resolutions in about 2 hours. This process is not interactive; however, the process is automated. As each movie image panel is rendered, it is automatically written over the network to the disks of the PC that will later be used to display it. The rendering process is coordinated by a globally accessible mySQL database.

5. X-RAY DIFFRACTION CRYSTALLOGRAPHY

X-Ray diffraction crystallography is a technique to provide detailed structures of molecules of atoms in the chemistry and biochemistry domain. A case study taken from X-Ray Diffraction Crystallography has been used to demonstrate and test the framework [9]. The application requires the transformation of a small data file into a ball and stick representation of a molecule and an iso-surfacing of electric field strength within the molecule.

The proposed architecture [10] allows peer-to-peer communication between Grid services that can be distributed across many different locations. This approach differs from other Grid visualization initiatives which operate in a client/server manner. The Peer-to-peer communication model theoretically reduces the communication overhead by 50% compared to the client/server model as traffic is transferred between services and not via a server, which

would result in information being transmitted twice over. The results were generated using IRIS Explorer toolkit and a graphics package.

6. VISUALIZATION OF THE SPEED OF POLLUTION FROM A CHIMNEY

In [11], a Grid-enabled version of IRIS Explorer was developed under gVis project (funded by the UK e-science programme), in which Grid middleware is incorporated within the internals of IRIS Explorer, allowing it to be run securely in a distributed processing environment. The prime instance of IRIS Explorer runs as before on the desktop, providing a library of modules and a workspace in which these modules can be connected into a dataflow pipeline. However, a secondary instance of IRIS Explorer can be allowed to be called up by another scientist, running on a remote Grid resource provided with a second library of routines, displayed on his/her desktop, from which he/she can select modules for inclusion in the workspace as part of the processing pipeline.

A grid-enabled IRIS Explorer application was demonstrated to visualize the spread of pollution from a chimney. By using the computing power provided by the Grid, scientists were able to simulate the event faster than real time with information about current weather conditions from the meteorologists. The meteorologists in return were able to collaboratively steer the simulation and visualize the results to predict the path of the pollutant in the shortest time so that local authorities can be informed and evacuation strategies can therefore be determined.

7. VISUALIZATION OF ADAPTIVE MESH REFINEMENT

VisPortal [12], a centralized system, which acts as a portal, was developed jointly by University of California, Davis and the Lawrence Berkeley National Laboratory (LBNL) to manage their resource and provide data visualization. The portal provides a web-based interface not just for exploring but also for encapsulating visualization data. Encapsulating the process lets the user reproduce the visualization results for validation or extend those results by continuing data exploration.

In [12, 13], visualization of Adaptive Mesh Refinement (AMR) data, numerical modeling involving meshes with huge ratios of scale, was performed by scientists in LBNL. A parallel

multiprocessor volume renderer was developed for AMR data. A client/ server architecture was created to enable the entire system to be accessible over the grid.

8. STRUCTURAL BIOLOGY/ MOLECULAR MODELING

Grid computing application in structural biology, as describes in [14], uses Electron Microscopy Tomography to investigate a structure of biological specimens over a wide range of sizes, from cellular structures to single macromolecules. Electron Microscopy provides structural information at low/medium/high resolution of macromolecules and also allows visualization of the macromolecules in their cellular native context.

In structure determination of biological specimens by EM, 3D reconstruction and data collection are needed. Thousands of projection volumes are combined to yield reconstructed volumes. Therefore Grid computing technology has the potential to reduce the heavy computational task.

As mentioned in [14], Grid computing can be the solution for structure determination of large viruses at high resolution where high resolution structure determination is computationally intensive and can help in the reconstruction of whole cells with sufficient resolution for visualization their molecular architecture where it is currently unapproachable.

The Grid implementation approach is based on the Weight Back Projection (WBP) method [14]. One slide of work is assigned to a node at time until all slices have been processed. The approach has a co-allocation strategy implemented so that available information from the batch scheduler is used to deploy the application in a grid environment. That sort of application-level scheduler is well suited to deal with the space-shared resources on the grid.

In Deep View system [5], an extension to OpenDX-MPI enables real-time concurrent visualization of the progress of distributed-parallel simulations. A secondary structures of proteins has been visualized interactivity running on GROMACS (Groningen Machine for Chemical Solutions), an open-source molecular dynamics simulation package (http://www.gromacs.org).

OpenDx-MPI allows the preservation of the distribution of data in a parallel simulation through the visualization process. The visualization process can take place in situ on the nodes of the distributed system on which the simulation takes place or by using the system's high-

performance intercommunications to transfer the data in a parallel to dedicated visualization nodes within the same distributed system.

9. COMPUTATIONAL FLUID DYNAMIC (CFD)

Real-time interactive visualization of computational fluid dynamic (CFD) simulations was performed to study two open problems (reflection and focusing of weak shock waves) [15] in compressible fluid dynamics.

A distributed application was built to allow the simulation and visualization programs to run on different machines connected via the internet and it is possible to connect to/disconnect from the ongoing simulation. This tightly coupled simulation-visualization framework system supports interactive visualization of “live” simulation data and it allows computational steering of the simulation by manipulating parameters to increase solution accuracy of the simulation.

10. COLLABORATIVE VISUALIZATION

Collaborative visualization [16, 17] of dataset between groups of trusted investigators was demonstrated using ICENI, the Imperial college e-science Networked Infrastructure, grid middleware system.

The basic ICENI graphics components are based on the Visualization Toolkit (VTK) and provide independent renderings of the data being exported. The collaborative users may view different data sets or different renderings of the same data set, but these will not be synchronized to each other. Additional components have therefore been developed to provide synchronization needs. The multiple steering and visualization components can be configured to provide collaborative interaction sessions between groups of trusted investigators. The Access Grid, open-source video conferencing tool, uses IP multicasting technology to efficiently transmit/receive audio and video streams over IP networks.

11. VISUALIZATION OF MULTI-PHYSICS SIMULATION

Japan Atomic Energy Research Institution [18] presented a real-time visualization for a multi-physics simulation with coupled compressible fluid (salve) and thermal conduction (master). The experiment was to investigate the dynamics of the compressible fluid confined in a closed cube

with 0.6-meter sides, inside a copper cubic vessel with 1-meter sides. Parallel tracking steering (Patras)/ITBL visualization system was used to carry out the visualization and Application Visualization System (AVS)/ITBL reads and visualized the stored data on the remote computers connected to the ITBL environment.

12. VOLUMETRIC IMAGE PROCESSING

Interactive visualization of 3D dataset is able to bring out the features of interest to user. However, Visualization of 3D data produced by tomographic imaging systems often requires post-processing to remove noise and enhance features of interest. An interactive grayscale morphology [19] of a “CT-Head” dataset (256x256x113) was performed to extract the brain area by two clusters of 9 x dual-processor Intel Xeon 1.7GHz, 1 GB RAM machines. A divide and conquer strategy was implemented where the grid computation space was divided into sub-volumes and dispatched to the nodes of the two clusters for both computation and volume rendering.

13. VECTOR FIELD VISUALIZATION

Vector-Field Visualization [19] is to illustrate fine changes in vector direction across voxels of large volumes. Line-Integral Convolution (LIC) technique is used in vector-field visualization and consists of blurring a noise function along a vector field to illustrate direction. Phase-shifted filter kernels were applied to generate time-varying 3D LIC volumes of 3D vector field data as an animation.

2.3 REMOTE ACCESS RENDER FARM

A render farm can be built by harnessing distributed computing resources and therefore this has been an alternative inexpensive solution for animators to handle their time consuming rendering tasks. A render farm is a cluster of interconnected computers which are used for rendering computer generated imagery. However, there are 4 basic concerns about building and having a render farm. 1) Physical space. Having a rack of 200 computers/blade servers networked together in the same location requires significant physical storage space. For small studios, this is infeasible. 2) Energy resources. Higher electrical power consumption is required for powering

the render farms and for maintaining a constant cool temperature in the server room. 3) Human resources. Hiring full time technicians/engineers for maintaining the servers and network. 4) Maintenance. Periodic upgrading of the render farm hardware and software.

There are commercial companies that offer on-demand computer animation online render farm services. Examples are, 1) Render Nation [20] in UK – 216 nodes with FTP with no web-interface. 2) Remote Render [21] - Costa Rica 3) render farm [22] in UK – 85 nodes. 4) Render now [23] - FTP with no web interface. 5) Render Core [24] – 1000 nodes, RenMan client with FTP access but no web interface. 6) ResPower [25] – the only commercial provider that supports Blender rendering. 7) Render Rocket [26] in USA. 8) Render-IT [27] - UK based render farm with 82 nodes.

2.4 SUMMARY

The main Grid-based visualization applications have been introduced. Visualization with Grid computing solution has provided visualization tasks with an improvement in computational time and previously impossible computation tasks of huge scale can be done under the framework of Grid Computing. Some commercial online rendering services have also been introduced. In next chapter, using grid computing technology to assist computer animation rendering will be discussed.

CHAPTER 3

GRID-BASED COMPUTER ANIMATION RENDERING FRAMEWORK

3.1 INTRODUCTION

In previous chapter, we have introduced how Grid computing has been employed in various different scientific research applications. Grid computing technology can also be used in the animation rendering domain. In this chapter, we will propose a Grid-based computer animation rendering framework.

Computer animation is a process where a series of still images is assembled and shown sequentially to generate an illusion of a continuous motion effect. Each image, or frame, in the animation sequence should blend seamlessly with other images to create smooth and continuous motion that flow through time.

Rendering is the process of converting 3D geometric models into graphics images. Rendering of an animated scene is a very time consuming task since thousands of frames are needed to create an animation. Rendering of a single frame in professional animation normally takes about several hours.

A common method to save rendering time is to reduce the scene complexity but this might compromise the quality of the final animation scene. Therefore, animators often have to find a balance between the quality of the scene and the production time. However in the recent years, animators were able to render highly complex 3D models for creating their animation sequences by using high performance networked computers.

Harnessing distributed computing resources to create a so-called render farm has therefore been a solution for animators to handle their time consuming rendering tasks. A render farm is a cluster of interconnected computers which are used for rendering computer generated imagery. There are two types of rendering methods: network rendering and distributed (split-frame) rendering. In network rendering, the images can be rendered in parallel, as each frame can be calculated independently of the others. In that case, the main communication between processors is used for uploading the initial models and textures and downloading the finished images. In distributed (split-frame) rendering, each frame is divided into tiles which are rendered in parallel.

Quite often small animation studios build their own render farms. These render farms usually consist of about 20 interconnected computers. The network connection is simple: with multiple switches/hubs and intranet cables connecting off-the-shelves computers. Quite often, the animators' computers are added to the render farm after office hours. Commercial render farm managing software is used to support the transaction and management of the rendering tasks between the database server and the networked computers. Small render farms allow for solving only relatively simple animation tasks, and their further extension is limited due to the budget and space constrains.

There are also commercial render farms available online as mentioned in previous chapter. The popular examples are RenderCore [24], ResPower [25] and Render-IT [26]. RenderCore (USA) and ResPower (USA) are commercial online render farms each with about 1000 computing nodes. RenderCore charges about USD$132 with ResPower charges about USD$213 for rendering 900 frames using *Mental ray*. This job takes about 4.0 minutes while on a single 3.2GHz computer it takes 3 minutes to render only one frame. However, this timing excludes the uploading of the 900 frames and downloading of the rendered images. Render-IT (UK) is a commercial online render farm with about 82 computing nodes.

The rendering time can be further reduced by having more computers in a cluster. However, keeping hundreds of computers in the same location requires a significant physical space. This limitation can be removed by linking up clusters of computers across different areas. This solution has brought up a new paradigm of computing called Grid computing. Computational Grid (or just Grid) can be defined as an infrastructure with interconnected nodes of resources. A node is an access point of communication or computation. The resources could be databases, application server, networks and storage devices. The function of a Grid is to enable sharing of geographically distributed heterogeneous resources between interconnected nodes. Grid has benefited many researchers and institutions in their visualization tasks such as earthquake simulation [7], visualizing speed of pollution for determining evacuation strategies [11], molecules modeling [14], real time visualization of multi-physics simulation [18] and volumetric image processing [19].

Most render farms are considered to be a cluster rather than a Grid as the computers are situated within the same location and with a single administration. For example, AXYZ Animation, Inc. [28] based in Toronto, Canada, produces digital special effects for commercials, television shows, and films. AXYZ uses SUN Grid Engine software [29] to harness and manage all computing resources in the company [30] for their rendering jobs. With the second processor of the animators' dual-processors computers specially dedicated to the Grid for all days rendering jobs, there was no impact on the animators' computers performance as they do their normal work. In another example, Ringling School of Art and Design [31], about 400 Apple Power Mac G4's and G5's located in 13 computer laboratories are harnessed together to form a render farm using Apple's XGrid [32] software. Although Grid services, such as security, data management, and resources management, were involved in these Grid implementations, they do not constitute a real Grid, because only local resources are used for rendering and all the computing resources are under a single administration.

The commonly available commercial render farm software are *EnFuzion* from Axceleon [33], Grid MP from United Devices [34], *Smedge* from Uberware [35], *Combustion* from Discreet [36], *mental ray* from mental images [37], and *RenderMan* [38, 39] from Pixar [40].

Commonly used *mental ray* provides a network and distributed rendering ability on a network. However, it requires a 'hosts' file (rayhosts) containing all the IP addresses of the rendering

machines. This might complicate using *mental ray* on the Grid since normally only the head node IP addresses of the remote clusters are available. Furthermore, *mental ray* needs to be installed on all the rendering machines and every node requires a *mental ray* license, which may result in a quite high cost of the rendering Grid.

3.2 PROPOSED GRID-BASED RENDERING FRAMEWORK.

Figure 3.1 shows a general framework of the proposed Grid rendering system. There can be two different approaches to building the Grid-based rendering system under this general framework. One approach is to set up a portal though which a user can submit a rendering job on-demand and download the rendered images from the remote server at a later time. A portal can be considered local to the rendering cluster, when it is accessed from the same location, or global, when remote clusters of computers are used. Another approach is to submit the job to the Grid through a GUI plug-in within the 3D modeling software environment where every rendered image will be sent back individually to the client concurrently.

In networking communication, there are 3 main factors which have to be considered, as mentioned in [41]: latency, bandwidth and contention. Latency is the time required to set up a communication operation, irrespective of the amount of data to be transmitted. Bandwidth is the amount of data which can be transmitted over a channel per unit time. Contention occurs when multiple processors are trying to route data through the same segment of the network at the same time and there is insufficient bandwidth to support the aggregate demand.

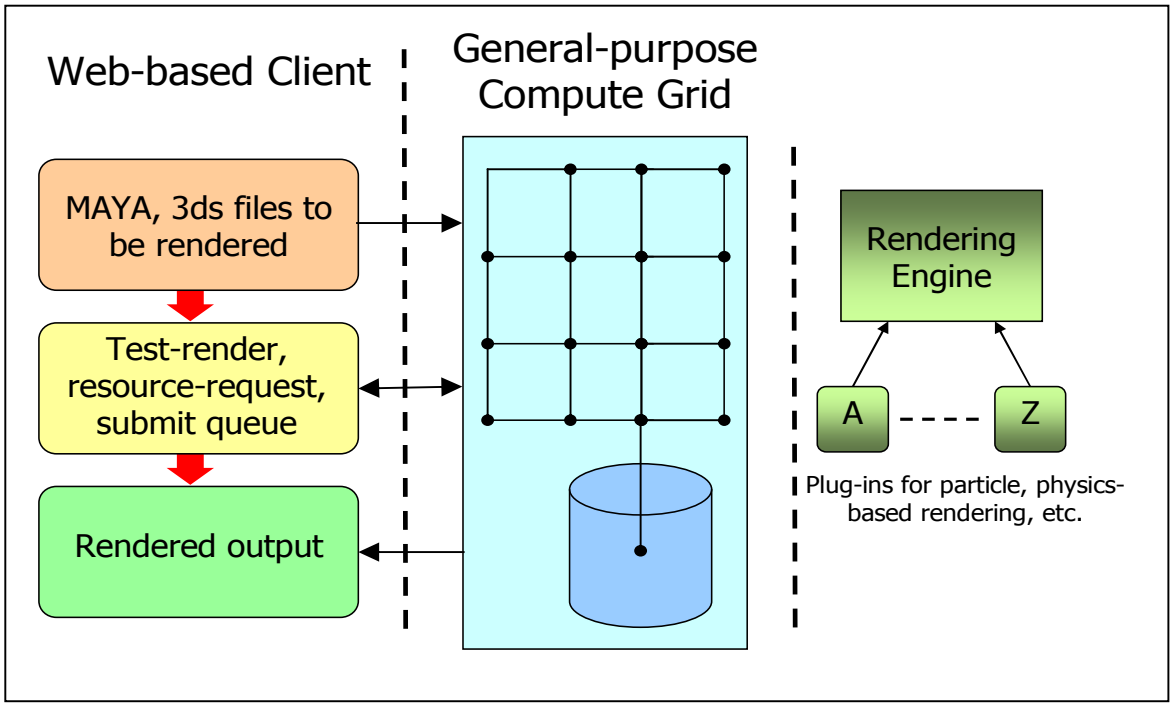

FIG 3.1 FRAMEWORK OF THE GRID RENDERING SYSTEM

Therefore, the time to send data from one processor to another one can be expressed as:

$$t_{comm} = t_{latency} + t_{transfer} + t_{delay}$$

where the total communication time, t_{comm}, is the sum of the latency ($t_{latency}$), data transfer time ($t_{transfer}$), and contention delay (t_{delay}). Latency can be further represented by the following sum of three components:

$$t_{latency} = t_{send} + t_{route} + t_{receive}$$

where, t_{send} is the time to initiate a transfer, t_{route} is the latency through the network, and $t_{receive}$ is the time to receive the data.

Most often, results of the scientific computations have to be downloaded from the grid to the local workstations before starting any rendering jobs. Downloading large data from the grid may require a substantial amount of time, and processing these data on a local workstation with limited computation power may be difficult [42].

Since animation sequences contain thousand of frames, transferring them over the network results in a long transfer time $t_{transfer}$. Therefore in our framework, a novel compression algorithm (filed for patent in Singapore) has been designed to solve the problem of transferring large 3D animation scene files. This compressor has been integrated with this Grid rendering framework.

When building the Grid rendering framework, the main emphasis is to focus on the friendly usability, security and stability of the framework. Our framework is customized to provide a simple, friendly, secure, and optimized Grid-based rendering environment for animators to submit and handle their rendering tasks either via a portal or a GUI plug-in in the modelling tool.

For submission of jobs to the Grid rendering system a portal is proposed (Figure 3.2). This portal is also an entry point for accessing the web services, meta-scheduler, Data Service (DS) and Execution Service Container (ESC) as shown in Figure 3.3. We compress the original scene representation files before uploading them to the rendering servers via the portal.

In the portal implementation, all computers in the Grid were installed with Globus Toolkit [43, 44]. Globus also provides Globus Security Infrastructure (GSI) core service and manages accesses and authorization using a public key infrastructure based on X.509.v3 and Secure Socket Layer (SSL) Protocol/Transport Layer Security (TLS) protocols for authentications.

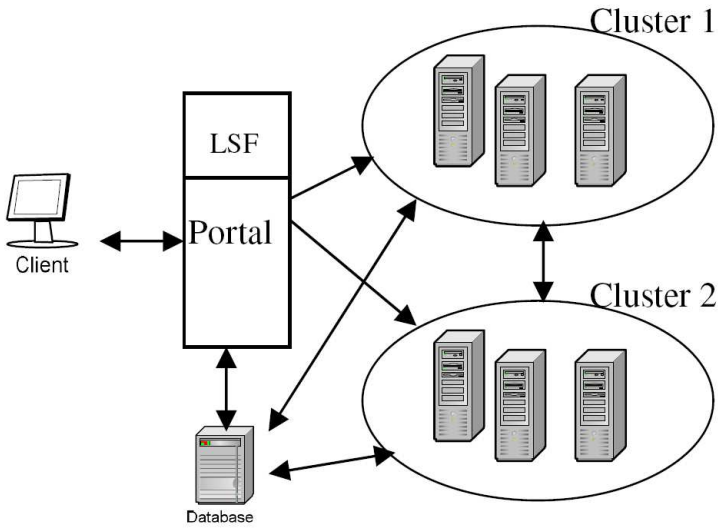

FIG 3.2 Using a portal for rendering jobs submission. Client will have to upload the animation frame sequences to the portal. The LSF (Load Sharing Facility) meta-scheduler has the ability to dynamically manage and distribute rendering jobs to the clusters for rendering and it is one of the job scheduler mechanisms supported by GRAM (Grid Resource Allocation Manager), a component of the Globus Toolkit. Each individual cluster has a local-scheduler (Sun Grid Engine) for managing the distribution of the rendering tasks. In case, if any job has failed to complete, it will be detected by the scheduler and re-submitted to other machines. LSF scheduling will not affect the final quality of rendering as the actual rendering are done by using Mental Ray or Pixie renderer. The rendered images can be downloaded from the database through the portal.

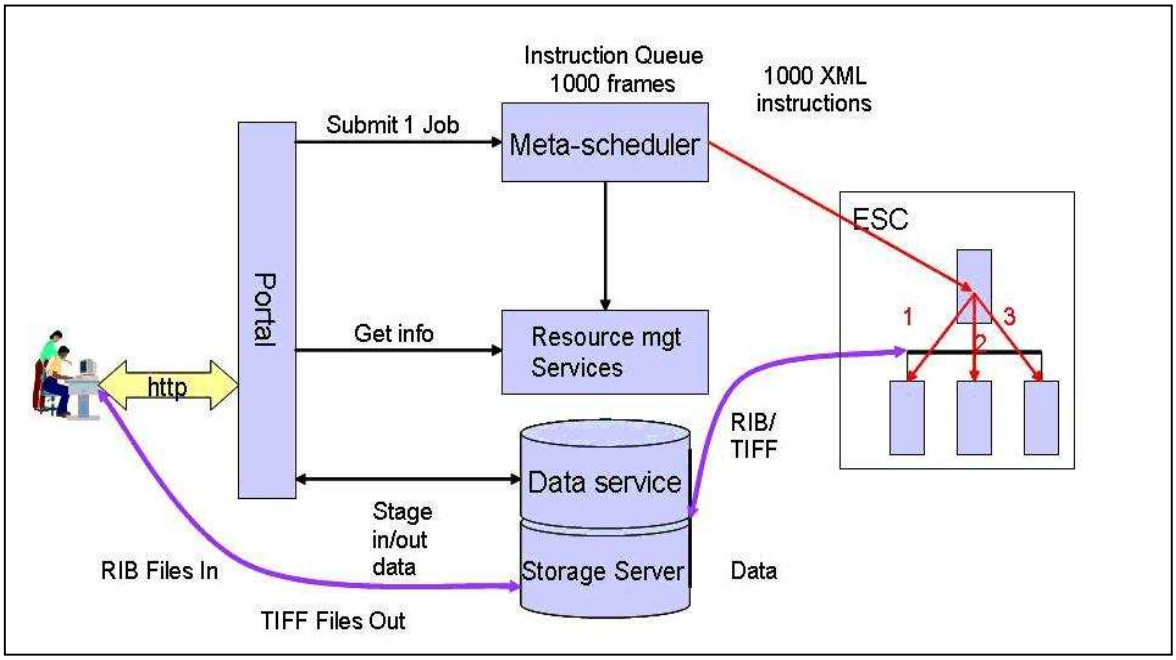

FIG 3.3 PORTAL RENDERING APPLICATION FRAMEWORK

The meta-scheduler resides on top of the local job schedulers, which manage computational resources for individual computer clusters and high performance computers. The meta-scheduler coordinates computing resource management across the collection of computer clusters. The meta-scheduler provides central job and resource management, enabling users to submit jobs without requiring knowledge of the Grid configuration and status. The meta-scheduler coordinates with all local schedulers to understand the current system status, and makes scheduling decisions when job submission is received.

Execution Service Container (ESC) encapsulates the local scheduler of the computer cluster and provides a Web service interface to access the computing resources. ESC has different adapters interacting with various local resource managers, such as Portable Batch System (PBS), Load Sharing Facility (LSF) and Sun Grid Engine (SGE). ESC invokes and manages execution of the applications deployed on the associated resource when a computation request is received.

Data Service (DS) runs at the portal server and it automates data transfer between the portal server and ESCs (Stage-in information to render and Stage-out rendered images to storage server) using data transfer protocols (GridFTP and FTP). For each job submission, the meta-scheduler coordinates DS and the associated ESC to complete the input and output data transfer. Both ESC and DS interact with the meta-scheduler scheduled by a Grid compliant protocol.

3.3 SUMMARY

In this thesis, a Grid-based rendering framework has been presented for on-demand animation sequence rendering. The proposed framework provides two methods for accessing the Grid: direct rendering and rendering through a portal access. Direct rendering allows for an immediate returning of the rendered images where the images can be checked and any fine-tuning of the images can be performed by terminating the current rendering process and re-sending the animation sequences frame back to the rendering Grid again. The Portal allows for an easy access to the render Grid and for sending batches of animation sequences frames. The returned images are stored in the storage server for later retrieval. However, scene representation files exported from 3D modelling applications to the rendering software formats (Renderman (.rib) and mental images (.mi)) often result in gigabytes of data size and uploading these gigabytes data

to the Grid is technically infeasible. Therefore, in the next chapter, a novel loseless 3D compression method will be proposed to solve the existing problem of transferring gigabytes of scene representation files.

CHAPTER 4

EFFICIENT COMPRESSION OF THE COMPUTER ANIMATION FILES FOR RENDERING AT MEDIA GRID

4.1 INTRODUCTION

In previous chapter, a computer animation rendering framework has been proposed. However, there exists a problem of transferring gigabytes of animation scene files to the Grid for rendering. In this chapter, we propose a novel data compression algorithm to provide a solution to this problem.

In computer animation, scene representation files exported from 3D modelling applications to the rendering software formats often result in gigabytes of data. Compression of data prior to sending them for remote visualization and subsequently decompression at the local host end was suggested in [45]. Another approach to solving the bandwidth problem is to use caching

strategies as proposed in [46] to improve the performance by avoiding redundant data transfer. Caching prevents sending gigabytes of data, but it requires bi-directional communication which is often not available. For example, for web-based grid applications, data is usually transferred using HTTP protocol [47], which does not allow arbitrary information to be sent back during the transfer. An alternative strategy is to perform data compression prior to sending them for remote visualisation and subsequently decompression at the local host end, as it was suggested in [45]. Commonly available compression tools such as *Winzip* and *gzip* can be used for compressing 3D scene files but they compress files individually and not able to take advantage of the redundancy between files. 7z and RAR formats offer a solid compression method for data compression of multiple files, where all the compressed files are concatenated and treated as a single data block. However, the internal structure of 3D animation scene files is unknown to them and they are unable to utilize the structure to obtain optimized compression ratio hence motivating further research on lossless data compression algorithm for 3D animation scene files.

In this thesis we propose an efficient lossless compression algorithm devised for transferring gigabytes of 3D scene representation files. We illustrate how the algorithm works within a developed framework for Grid rendering services based on using *RenderMan* and *mental images* data formats.

The chapter is organised as follows. Section 2 surveys the existing compression algorithms. Section 3 describes a novel algorithm devised for compressing 3D animation scenes. Section 4 summarises this chapter.

4.2 DATA COMPRESSION FOR ANIMATION SCENE FILES

There are many known methods for data compression. Though devised for different applications, they all are based on the same principle of compressing data by removing redundancy. These methods can be classified into two groups: statistical-based and dictionary-based.

4.2.1 STATISTICAL-BASED METHODS

In statistical-based compression methods, most real-world data have statistical redundancy [44] where the same type of source material is used consistency. For example in English text, letter '*e*' is much more common than letter '*z*'. Allocating short codes to frequently used letters and allocating long codes to infrequently used letters is often performed to reduce the amount of redundancy in the data.

Popularly known Huffman coding [49, 50] is designed for data source having known statistics. Huffman coding first counts occurrences of data items and builds the binary Huffman-tree. New binary representation (bit pattern) for each data item is computed. The data is re-coded with the new bit-patterns thus reducing the original data size. Huffman coding provides the advantages of optimal representation of characters (saves space) and fast decoding using Huffman-tree. However, Huffman coding has some disadvantages. Huffman-tree must be saved with the encoded data for decoding hence small files can become bigger. Also, two procedures are needed for encoding: one reads the file, counts the characters and builds the Huffman-tree, while the other one reads the file again and encodes each character where the process can slow down.

Adaptive Huffman coding [51] has been designed where the coding can be done on-the-fly. Starting with an empty tree, the first input symbol is written to the output stream in its uncompressed form. The symbol is then added to the Huffman tree and a code is assigned to it. If the same symbol is subsequently encountered, its code is written on the stream and its frequency is incremented. The tree will be checked and updated if it is no longer a Huffman tree.

4.2.2 DICTIONARY-BASED COMPRESSION METHODS

In dictionary-based compression methods, data is compressed by replacing substrings of the data by pointers to matching strings in the dictionary. The dictionary holds strings of symbols and it may be static (no deletions of string, but sometimes allow additions) or dynamic (adaptive) of strings where the previously held strings are found in the input stream, allowing for additions and deletions of strings as new input is being read. A word is read from the input stream and the dictionary is searched. If a match is found, an index to the dictionary is written into the output

stream. Otherwise, the uncompressed word itself is written. LZ77, LZ78 and LZW are some of the dictionary-based compression methods.

LZ77 is a dictionary based algorithm [52, 53] that addresses byte sequences from former contents instead of the original data. LZ77 is based on a sliding window that uses a part of the past input stream as the dictionary. The encoder maintains a window to the input stream and shifts the input in that window from right to left as strings of symbols are being encoded.

LZ78 [54] is based on a dictionary that will be created dynamically at runtime and no combination of the address and sequence length is used as in LZ77 algorithm. Instead only the index to the dictionary is stored. While LZ78 algorithm attempts to work on future data, LZ77 algorithm works on past data. Input data is scanned and matched against the dynamic dictionary. If the input symbol is found in the dictionary, the next symbol is read and concatenated with the first one to form a two-symbol string that the encoder tries to locate in the dictionary then. As long as the strings are found in the dictionary, more symbols are read and concatenated to the string. If at a certain point the string is not found in the dictionary, the encoder will add it to the last symbol of the string (the one that caused the search to fail) as its second field. LZ78 has an advantage over LZ77 as nothing is ever deleted from the dictionary therefore future strings can be compressed even by strings seen in the past. However, the dictionary tends to grow fast and fill up the entire available memory.

LZW method [55] starts by initialising the dictionary with the first 256 characters before any data is received. Because the dictionary is initialised, the next input character will always be found in the dictionary. It reads 8 bits of the data at a time (e.g., '*a*', '*h*', etc.) and encodes the data as the number that represents its index in the dictionary. When it comes across a new substring (e.g, "*th*"), it adds it to the dictionary. When it comes across a substring it has already seen, it just reads in a new character and concatenates it with the current string to get a new substring. The next time LZW revisits a substring, it will be encoded using a single number.

The most widespread and popular format is the ZIP file format that provides compression and also acts as an archiver by storing many files in a single output file. It uses the “deflation” algorithm, which uses a variation of LZ77 combined with the static Huffman.

4.2.3 LOSSLESS AND LOSSY COMPRESSION

In data compression, there are lossless and lossy compression. Lossless data compression allows the exact original data to be reconstructed from the compressed data [56]. Lossless compression is used when it is required that the original and the decompressed data are to be identical. Typical examples are executable programs, source codes and 3D scene representation files. Some image file formats, notably PNG Portable Network Graphics, use only lossless compression while TIFF format may use both lossless and lossy methods.

Lossy methods are most often used for compressing sound, images or videos where some loss of resolution or quality is often undetectable and acceptable [57]. Lossy compression algorithms introduce relatively minor differences and represent the picture, video, or audio using fewer bits.

The human eye is fairly good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation. A stream can be compressed by employing quantisation technique [58] where the amount of information in the high frequency components can be reduced by simply dividing each component in the frequency domain by a constant for that component before rounding to the nearest integer. This is the main lossy operation in the whole process. As a result of this, it is typically the case that many of the higher frequency components are rounded to zero, and many of the rest become small positive or negative numbers. Examples of this are the DCT data quantisation in JPEG [59] and DWT data quantisation in JPEG 2000 [59].

The psycho-acoustic model can describe which parts of a given digital audio signal can be removed (or compressed) safely without significant losses in the quality of the sound [60]. Flaws caused by lossy compression, that are noticeable to the human eye or ear, are known as compression artifacts.

When a user acquires a lossy-compressed file, (for example, to reduce the download time) the retrieved file can be quite different from the original at the binary level while being indistinguishable to the human ear or eye for most practical purposes as human eye can see only certain frequencies of light.

4.3 DATA COMPRESSION ALGORITHM

A novel efficient lossless 3D compression technique (filed for patent in Singapore) was devised for encoding the converted scene representation files (Renderman (RIB) and mental images (mi). Our understanding in frame to frame coherence (similarity between consecutive frames) in animation allows us to create a novel method of compression that focuses on the 3D geometrical objects in the scene files that occupy the majority of the file size. In our compression, duplicated 3D geometrical objects were extracted from the files, hashed and formed into a dictionary. All duplicated 3D geometrical objects will have the same hash value. These small hash values will replace the large 3D geometrical objects therefore reducing the original total file size. Hash values are then sent across the network instead of the duplicated 3D geometrical objects. By using these hashed values, the 3D geometric objects are read from the hash table (dictionary) when the reconstruction takes place. The decompression procedure will be performed at the portal side to form the compressed file to the original rendering files.

RIB standard is capable to instantiate (reference) objects that were defined earlier in the same file. However, it is necessary to have separate files for each frame for rendering on different machines. Instantiation does not work between files, and, therefore is not useful for our framework. Besides, the instantiation is limited to geometry, while our compression algorithm can work with any data.

Currently animators have to manually include textures and shaders required by the scene. A missing texture might cause the scene not to be rendered properly and will need to be sent for rendering again. This may affect the budget and the delivery schedule of the animators. Our compression procedure scans for associated textures and shaders that are required by the scene representation files for the rendering and extract them from the original location to be bundled together with the compressed scene file as a single zipped package for transferring to the portal across the network.

Meanwhile *mental ray* offers a good compression ratio during conversion from 3D modeling software to scene representation files (.mi) but in order to achieve that, animators have to select an option in the 3D modeling software that converts all animation frames to a single scene representation file. However, the selected frames must be in running sequences. Animators also

have to manually identify the associated files (textures and shaders) and use commercial compression software to package them into a single zipped file for remote rendering. There are some cases where animators have to re-render many non-sequential blocks with small frame sequences. These non-sequential blocks will be converted separately and therefore could not achieve the high compression ratio. By employing our compression method to these blocks of compressed rendering frames, these blocks can be compressed into a final compressed file for remote rendering. Furthermore, the final compressed file includes all the required textures and shaders that were not exported during the conversion stage therefore eliminating the necessary steps to locate them individually.

The proposed data compression algorithm does not support NURBS modeling directly. However, animators are able to use the 3D Maya built-in NURBS modeling function to work on their 3D models. The completed 3D models done using NURBS will be converted to either .rib file or .mi file where the file will be compressed by our proposed compression algorithm for rendering.

Our lossless compression and decompression method helps us to avoid redundant data transfer and allows for sending large dataset via the network without losing any data. Large converted scene files also can be stored at very small size in database and can be decompressed when needed. Our compression method allows such scene representation files to be compressed up to 98%. The general flow chart of the compression-decompression is shown in Figure 4.1.

A novel efficient lossless 3D compression method was devised for encoding the converted scene representation files. It compresses the files at the content level rather than at the low binary level as other compression methods do. The idea of the method is based on frame to frame coherence (similarity between consecutive frames) in animation. The method focuses on the 3D geometric objects in the scene files that occupy the majority of the file size and eliminates the redundant transfer of duplicated 3D geometric objects. We employed Message-Digest algorithm 5 (MD5) hashing function into our algorithm for hashing the 3D geometric objects text representation in the RIB/Mi files into sets of 32 digit hexadecimal numbers representation so as to reduce the scene file size.

The following steps describe the compression algorithm shown in Figure 4.2:

1) Geometry objects in scene representation files are scanned and the hash function of the geometry object data being computed sequentially.

2) Computed hash value of the first object is recorded into an emptied dictionary followed by the next sequential object which checked for duplication with the dictionary.

3) Non-duplicated hash value will be appended into the dictionary and the geometry objects being appended into a separate log file (compressed.log). Duplicated object will be replaced by the hash value and being appended into the log file.

Each set of object data have a unique hash code associated with it during compression. The unique hash code is substituted for all object data in the scene data which contained in a set of object data associated with that hash code enabling large reduction of the original total file size.

The process of scanning for object code is repeated until all object code has been detected and processed accordingly. Thereafter, the compression module automatically scans the scene data again in order to detect and extract texture/shader files. Whenever texture/shader file is encountered, it is extracted and bundled with the log file that contains the dictionary. Further compression is done using a compression software application, before being sent to the decompression module via the Internet/network through the web portal.

During decompression (forming back the original files), the dictionary provides a reference between the hash value and associated first encountered object data such that the first object data may be identified during the decompression of the compressed scene data. When the compressed scene data is being decompressed, whenever a hash value is detected in the compressed screen data, the dictionary is scanned in order to provide a reference to its associated first object data which should be substituted in place of all occurrences of the associated hash value in the compressed scene data. Signature encountered in the compressed file indicates a new scene file for each individual frame that has to be created for rendering. Figure 4.3 shows a flow chart of the decompression algorithm.

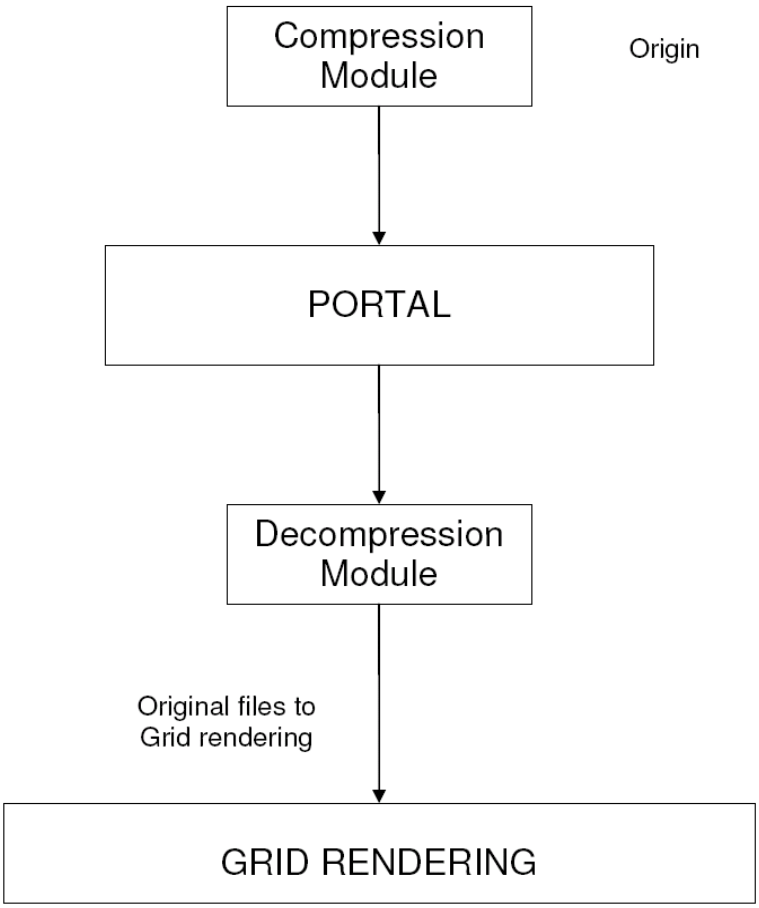

FIG 4.1 FLOW CHART OF THE COMPRESSION-DECOMPRESSION

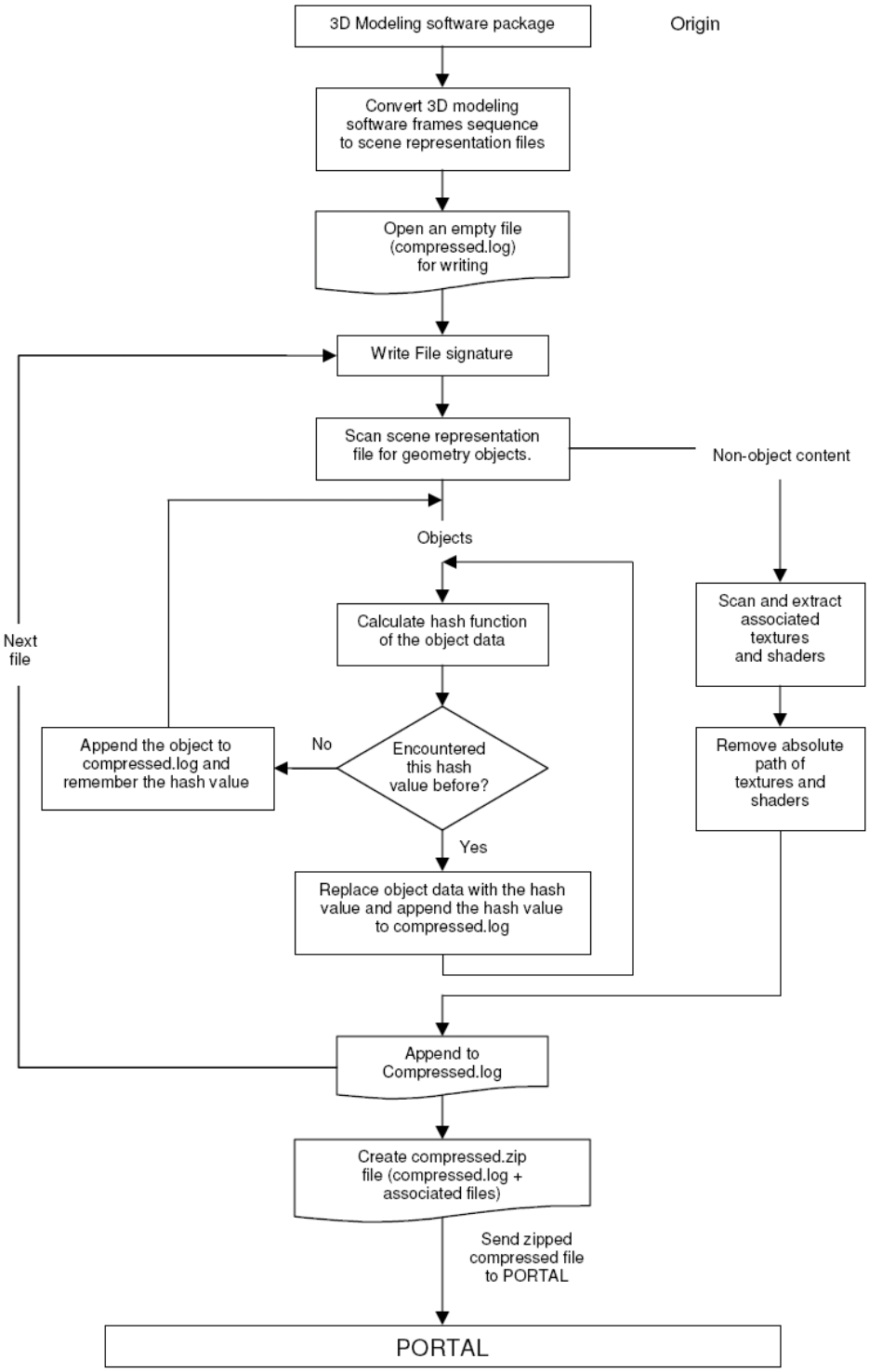

FIG 4.2 COMPRESSION FLOW CHART

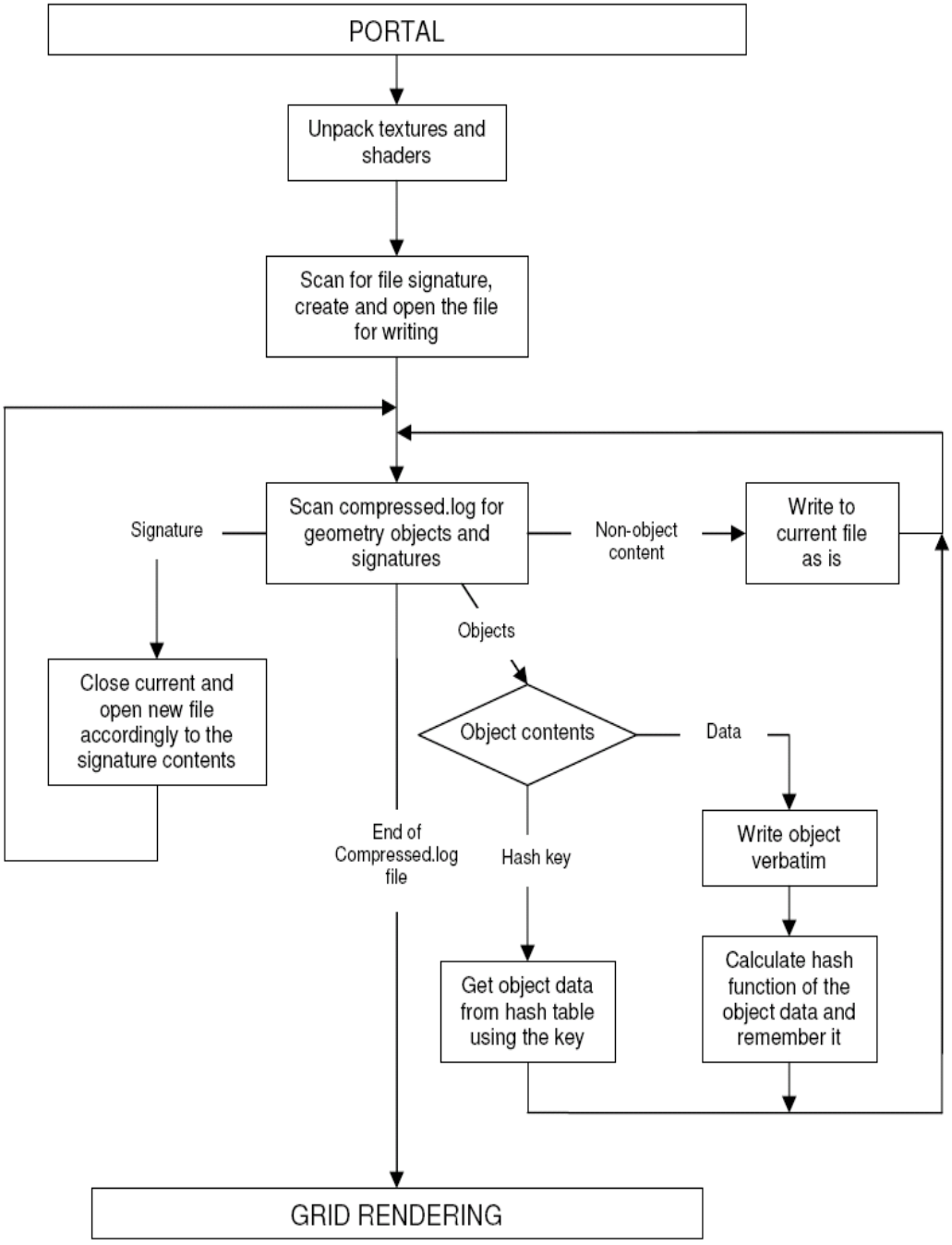

FIG 4.3 DECOMPRESSION FLOW CHART

It is also important to distinguish our method from the method called *diff* [61] wh*ich is also used* for compressing *RIB* and *mi* files.

There are principle differences between *diff* compression and our method. We make use of the internal structure of 3D scene description file and also the dictionary of 3D objects while *diff* uses universal Lowest Common Subsequence algorithm. The algorithm complexities are O(n) and O(n^2) respectively, where n is a file size.

There are also the following shortcomings of *diff* which were solved by our compression method:

1) *diff* does not have any knowledge about the structure of the 3D file. Therefore it has to use a general algorithm for finding common subsequence. With the regular size of data (more than 1 GB) this general approach becomes unfeasible. Our compression method knows exactly where to look for redundant data and therefore has a much faster compression speed (1 GB file compression with *diff* takes ~30 minutes, while with our compression method - ~5 minutes).

2) *diff* compares each frame with the first one, which means that any object that is not present in the first frame will not be compressed, even if it is found in several other frames as well. Our compression method detects all identical objects in all frames, hence the compression ratio is better.

3) *diff* will not handle several frames per one file. In practice, this is quite common condition, and our compressor handles it well.

4) *diff* compression method ignores all the resources referenced by the 3D scene. Our compression method tracks all the files necessary for the given scene and bundles them into the archive.

4.4 SUMMARY

In this chapter, a novel lossless 3D compression method has been proposed to solve the existing problem of transferring gigabytes of scene representation files. It compresses the files at the content level rather than at the binary level as other compression methods do. The idea of the method is based on frame to frame coherence (similarity between consecutive frames) in animation. The method focuses on the 3D geometric objects in the scene files that occupy the

majority of the file size and eliminates the redundant transfer of duplicated 3D geometric objects. The method was filed for patent in Singapore.

C H A P T E R 5

IMPLEMENTATION OF THE MEDIA GRID FRAMEWORK

5.1 INTRODUCTION

The described compression algorithm in the previous chapter has being used in the framework of the Media Grid rendering system which allows for two different ways of submitting jobs to the Grid-based rendering system: through a GUI plug-in within the 3D modelling software and through a web-portal. An open source Graphical User Interface (GUI) plug-in (*http://sourceforge.net/projects/liquidmaya/*) for the 3D modelling tool is used in this project. This plug-in converts the Maya scene to *RenderMan RIB* format and uses *Pixie* (*http://sourceforge.net/projects/pixie*) for rendering. The animator logs onto the Grid via the plug-in GUI, and the Grid identifies and authenticates the client (animator's computer). Upon authentication, the compressed package file from the client (animation scene representation files, associated textures and shaders) is sent to the meta-scheduler LSF (*http://www.platform.com/Products/Platform.LSF.Family*) via a Ruby script. Upon receiving the compressed package file, decompression takes place and the LSF manages and distributes all

jobs to the clusters and the meta-scheduler Sun Grid Engine (SGE) in each cluster distributes all the animation sequence frame jobs to the available rendering server (computer that performs the rendering) in its cluster.

This batch rendering process is designed to appear transparent to the animators who only click at the render icon to send the animation sequences to the Grid, and the rendered images would be stored in a database accessible for downloading by the animator. The animator is able to check the progress of the rendering and to preview the output rendered images.

The underlying framework in the portal works exactly the same as the direct rendering from Maya except that the portal allows animators to submit their animation scene representation files from any computer with or without 3D Maya installed. Animators just need to collect and submit the compressed animation package (*.rib*/*.mi* files, associated textures and shaders) for rendering via the portal as shown in Figure 5.1. Animators are able to check the progress of their rendering jobs by accessing their account as shown in Figure 5.2. Animators are able to collect their results via the portal as shown in Figure 5.3. The rendering time is the same for both direct rendering and portal rendering. Table 1 shows the time taken to rendering 100 frames using 60 nodes and a single node. The total time needed to render 100 frames using 60 nodes is 170 seconds compared to 6897 seconds to render 100 frames using a single node.

The portal is also an entry point for accessing the web services, meta-scheduler, Data Service (DS) and Execution Service Container (ESC). We compress the original scene representation files before uploading them to the rendering servers via the portal. In the portal implementation, LSF meta-scheduler resides on top of the local job schedulers to manage computational resources for individual computer clusters and high performance computers. The meta-scheduler coordinates computing resource management across the collection of computer clusters. LSF meta-scheduler provides central job and resource management, enabling users to submit jobs without requiring knowledge of the Grid configuration and status. The meta-scheduler coordinates with all local schedulers to understand the current system status, and makes scheduling decisions when job submission is received.

ESC encapsulates the local scheduler of the computer cluster and provides a Web service interface to access the computing resources. ESC has different adapters interacting with various

local resource managers, such as Portable Batch System (PBS), LSF and SGE. ESC invokes and manages execution of the applications deployed on the associated resource when a computation request is received.

DS runs at the portal server and it automates data transfer between the portal server and ESCs (Stage-in information to render and Stage-out rendered images to storage server) using data transfer protocols (GridFTP and FTP). For each job submission, the meta-scheduler coordinates DS and the associated ESC to complete the input and output data transfer. Both ESC and DS interact with the meta-scheduler scheduled by a Grid compliant protocol.

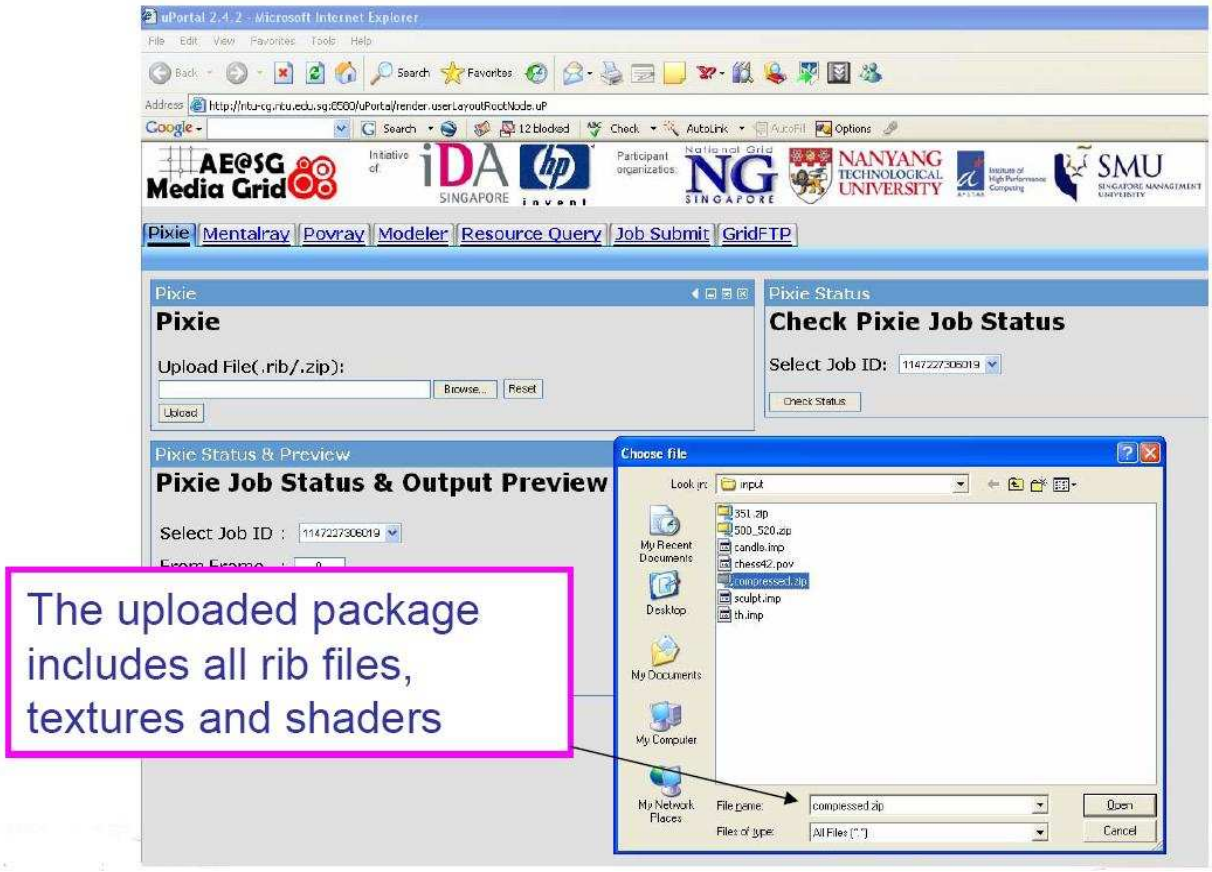

FIG 5.1 JOB SUBMISSION FROM THE PORTAL

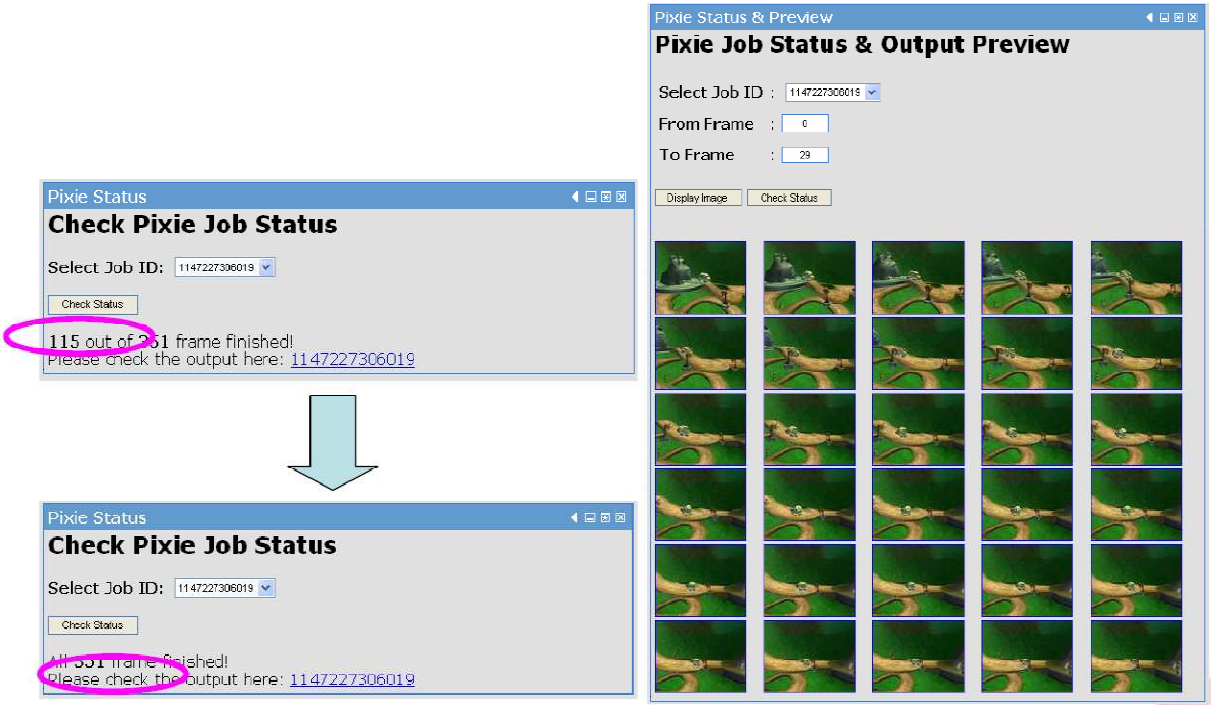

FIG 5.2 CHECKING STATUS FOR THE SUBMITTED JOB

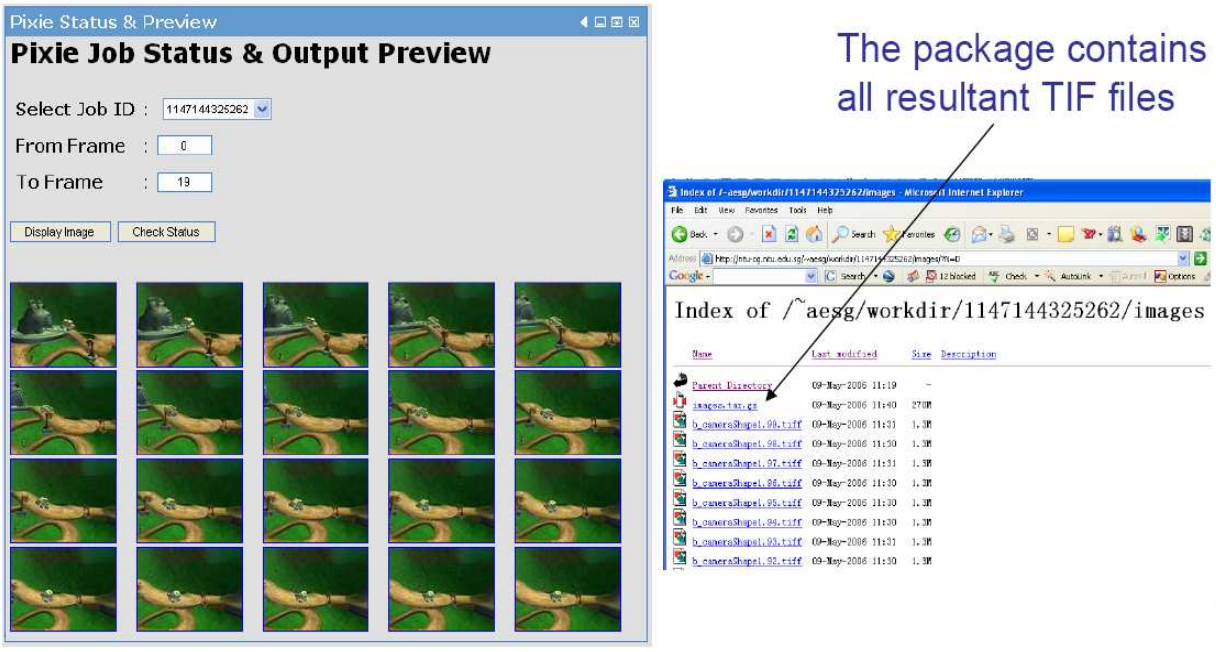

FIG 5.3 COLLECTING COMPLETED IMAGE FILES FROM THE PORTAL

TABLE 1: EXPERIMENTAL RESULTS FOR ROBOT ANIMATION SEQUENCE RENDERING; 100 FRAMES OF 1208 X 1024 RESOLUTIONS

Hardware involved (type and number of CPUs)	No. of frames (1280 x 1024)	Total Rendering time (hh:mm:ss)
1 x 40 nodes of Xeon 3.2GHz 1 x 20 nodes of Xeon 3.2GHz	100	00:02:50
One node of Xeon 3.2GHz	100	01:54:57

5.2. EXPERIMENTAL RESULTS

Within this framework, we implemented our compression algorithm for *RIB* and *mi* data formats (*RenderMan, https://renderman.pixar.com* and *mental images http://www.mentalimages.com*, respectively). Associated textures and shaders required by the scene representation files for the rendering were scanned and extracted from the original location to be bundled into a single file for transferring to the destination. The compression time is ~5.32 minutes. Decompression procedure was performed at the destination side to convert the compressed file to the original rendering files. The decompression time is ~5.20 minutes.

A Ruby script is used to send the animation sequence frames for rendering to 2 different clusters (1 x 40, 1 x 20) of total 60 remote computing nodes and the rendered images are returned to the animator in parallel. The rendered images are named according to their placement in the sequence in minutes and frames numbers for easy reference of frames for post-production or for video editing. For example, frame 90,000 represents the 50th minutes for a 30 frames/second animation video. Animators can just select <90000_50.tif> to <901800_50.tif> that represent the entire 50th minutes of the animation video.

From the results shown in Tables 2, 3 and 4, the rendering time has been greatly reduced by employing 2 different clusters of total 60 remote computing nodes for rendering.

TABLE 2: EXPERIMENTAL RESULTS FOR BUMBLE LAND ANIMATION RENDERING; 700 FRAMES OF 768 X 576 RESOLUTIONS

Hardware involved (type and number of CPUs)	No. of frames (768 x 576)	Rendering time (hh:mm:ss)
1 x 40 nodes of Xeon 3.2GHz 1 x 20 nodes of Xeon 3.2GHz	700	00:04:53
One node of Xeon 3.2GHz	700	03:53:33

TABLE 3: COMPRESSION RATIO FOR 100 FRAMES ROBOT ANIMATION SEQUENCE

No. of frames	Files size before compression	Files size after compression	Compression ratio
100	1,474.56 megabytes	38.43 megabytes	97.39%

TABLE 4: COMPRESSION RATIO FOR 700 FRAMES BUMBLE LAND ANIMATION SEQUENCE

No. of frames	Files size before compression	File size after compression	Compression ratio
700	4,618 megabytes	102.88 megabytes	97.77%

The rendering framework has been implemented and tested using two animation models: a robot that has 100 frames (3 seconds) as shown in Figure 5.4, and a bumble car environment with 700 frames as shown in Figure 5.5. Figure 5.6 shown several rendered robot animation frame sequences.

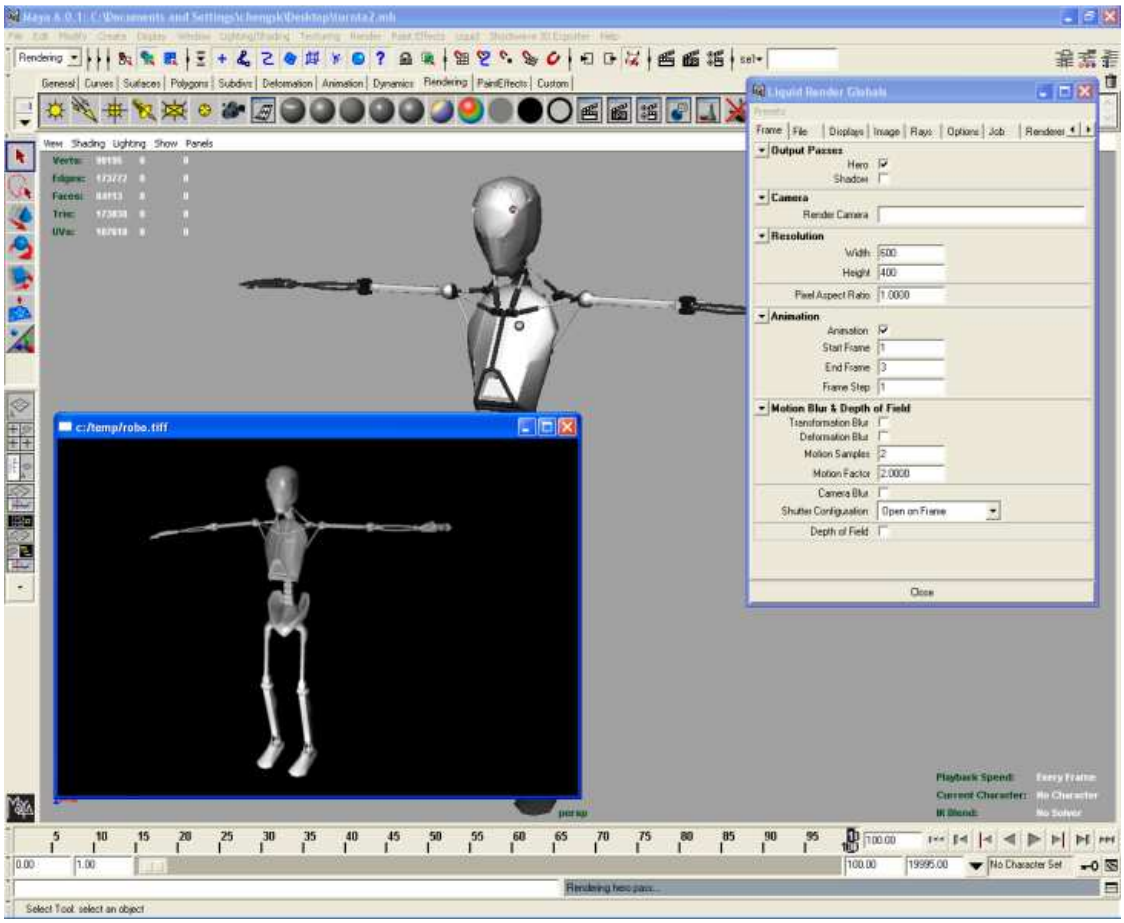

FIG 5.4 THE PLUG-IN TO MAYA 3D PROVIDES RENDERING ON-DEMAND. A SINGLE FRAME OF THE ROBOT ANIMATION SEQUENCE IS SHOWN AS ABOVE.

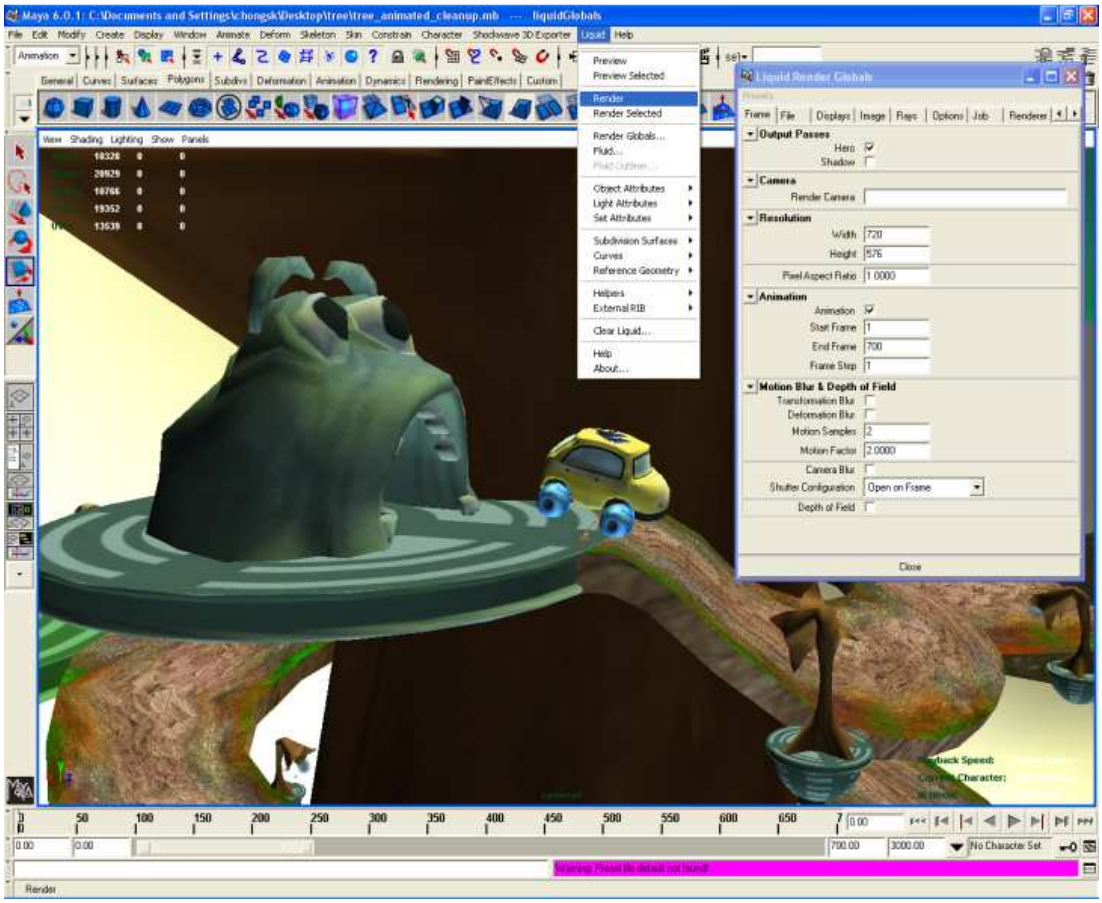

FIG 5.5 A SINGLE FRAME OF A 30 SECONDS BUMBLE LAND CAR ANIMATION. TOTAL 700 FRAMES.

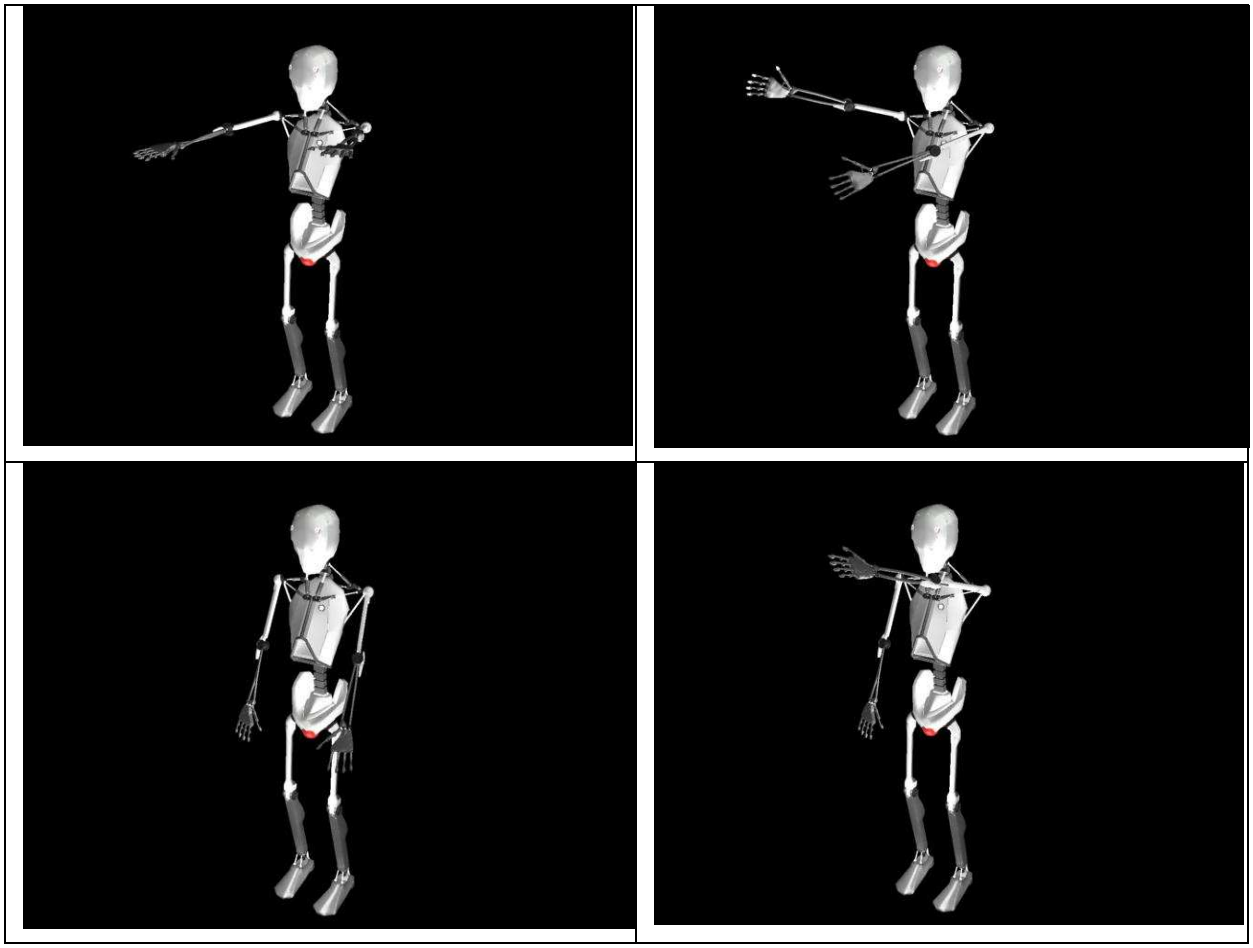

FIG 5.6 SEVERAL FRAMES OF A 3 SECONDS ROBOT ANIMATION. TOTAL 100 FRAMES.

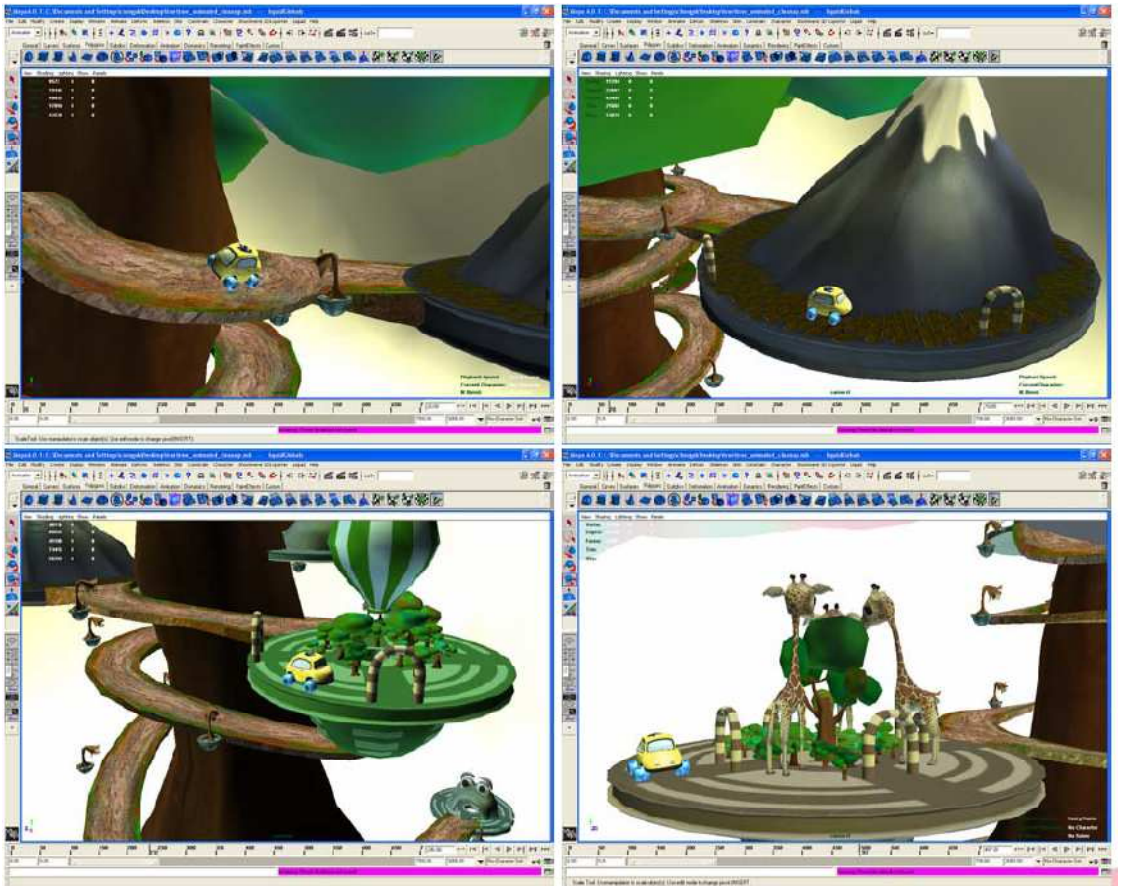

FIG 5.7 PART OF THE COMPLETE ANIMATION SCENE

5.3. SUMMARY

In this chapter, an implementation of the Grid rendering framework has been demonstrated. The two methods of submitting the scene files: direct rendering and rendering through a portal access have been achieved. Our data compression algorithm allows the system to achieve a ~98% compression rate. The total time required for the rendering has been reduced significantly with the usage of our proposed Grid-based rendering system.

CHAPTER 6

CONCLUSIONS AND FUTURE WORKS

This chapter summarizes the work presented in this thesis and points out directions for future work.

6.1 SUMMARY

In this project, a Grid-based rendering framework was developed for solving 3D computer animation rendering problem. 3D Computer animation rendering is a very computationally expensive task depending on the complexity of the modelling scenes which could take from a few minutes to a few hours for a single frame. Most small and middle animation studios have limited resources for their animation production and have to observe tight production deadlines. They were unable to accept multiple big projects due to the lack of computer resources therefore affecting their revenues.

The project aimed at developing an inexpensive, user-friendly web-based rendering system using open source software where computationally expensive computer animation rendering work can

be performed with ease and inexpensively by animators from small to middle scale 3D animation studios.

We have accessed our project's research and development niche with commercial on-demand 3D computer animation rendering services on the internet. Currently, all of them only allowed portal access for animation rendering. Animators have to upload their original 3D models created, for example, in 3D Maya or 3D Studio Max along with the associated textures and shaders for rendering through the internet. There exists a security issue to avoid unauthorised access to their works during the process of uploading to the internet. Furthermore, the impossibility of accessing the rendering resources directly from the 3D modelling software prevent the animators from fast checking and fine-tuning of their works during the process of modelling.

The proposed Grid-based rendering framework provides two methods for accessing the Grid: 1) direct rendering, and 2) rendering via a portal. Direct rendering allows for an immediate return of the rendered images where they can be checked and any fine-tuning of the images can be performed by terminating the current rendering process and re-sending the animation frame back to the rendering Grid. The Portal allows for an easy access to the render Grid and for sending batches of animation frames. The returned images are stored in the storage server for later retrieval.

A novel and efficient 3D data compression algorithm was devised during the course of the project and filed for a patent in Singapore. The data compression algorithm serves two very important purposes in supporting the Grid-based rendering framework:

1) To solve the existing problem of uploading gigabytes of scene representation files (RenderMan (.rib) and mental images (.mi)) to the framework for rendering. Scene representation files exported from 3D modelling applications to the rendering software formats often result in gigabytes of data size and uploading these gigabytes data to the Grid is technically infeasible. Our data compression algorithm achieved a ~98% compression rate as it compresses the files at the content level rather than at the binary level as other compression methods do.

2) To act as an encryption mechanism for files uploading. The compressed files, without the de-compressor at the server-side, will be meaningless to anyone who managed to obtain them.

6.2 PUBLICATIONS

We have tested the developed framework in educational institutions (Temasek Polytechnic and Nanyang Polytechnic) and published papers at international conferences. We have also filed for a patent for our 3D data compression algorithm in Singapore.

Paper published

1. Anthony Chong S.K., Konstantin Levinski and Alexei Sourin, **Interactive Grid-based Free-Form Shape Modeling**, pp 253-256, 6th IEEE International Symposium on Cluster Computing and the Grid 2006, Singapore.

2. Anthony Chong, Konstantin Levinski and Alexei Sourin**, Grid-based Computer Animation Rendering**, pp 39-47, 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (ACM Graphite), Kuala Lumpur, Malaysia, 2006.

3. Bu-Sung Lee, Alexei Sourin, Clement Chia Liang Tien, Chan Kai Yuan, Terence Hung, Quoc-Thuan Ho, Jie Wei, Danny Oh, Steven Miller, Junwei Zhang, Anthony Chong and K. Levinski, **Rendering-On-Demand Service Across Heterogeneous Grid Environment**, pp 91-103, GCA 2007 Proceedings of the 3rd International Workshop on Grid Computing and Applications, June 2007, Singapore.

Patent filed

1. Authors: Chong A, Sourin A, Lee F, Levinski K. A.
Patent filed "**A method and system for compressing and decompressing data**", NTU ITTO Reference PAT/036/06/06/SG. Singapore Patent Application No: 200607321-7.

6.3 FUTURE WORK

This thesis represents a major step towards the development of the proposed framework for Grid-based rendering system. Two future work directions have been identified:

1. 3D data compression algorithm optimization.

The current 3D data compression algorithm could be further improved and optimized in order to improve the speed of compression and decompression. This algorithm should also allow compression of other type of scene representation formats.

2. Grid rendering computing framework expansion

This can be done by developing various plug-ins to enable direct access to our portal for some of the common popular 3D modeling software such as SoftImage (AutoDesk), LightWave 3D (NewTek), Houdini (Side Effects Software) and open-source Blender.

APPENDIX A

REFERENCES

[1] George D., Veronika M. IBM Digital Media: Solution for the Media and Entertainment Industry. Building an animation and special effects studio from the ground up. *http://www.ibm.com/solutions/digitalmedia/doc/content/bin/files_15.pdf*

[2] D. M. Akbar Hussain, Abdul Q. K.R., Bhawani S. C. and Quintin G. 2008. How to Build an Open Source Render Farm Based on Desktop Grid Computing. *IMTIC 2008*, CCIS 20, pp. 268-278.

[3] The Centre for Parallel Computing (CPC) [3] at the University of Westminster, London (*http://rendering-portal.cpc.wmin.ac.uk*)

[4] Krishna P.C. Madhavan, Laura L. Arns, and Gary R. Bertoline. 2005. A Distributed Rendering Environment for Teaching Animation and Scientific Visualization. *IEEE COMPUTER GRAPHICS AND APPLICATIONS.* Vol. 25(5): 32 - 38

[5] James T. K., Peter D. K. Julia V., Greg Abram. Christopher J. M., Robert H. W., and Thomas J. 2002. Deep View: High-Resolution Reality. *IEEE COMPUTER GRAPHICS AND APPLICATIONS*. Vol. 22(3):12-15.

[6] Greg H., Mike H., Ng R., Randall F., Sean A., Peter D. K., James T. K. 2002. Chromium: A stream-processing framework for interactive rendering on clusters. *SPECIAL ISSUE: PROCEEDINGS OF ACM SIGGRAPH 2002*. Vol. 21(3): 693-702.

[7] Martin A., Peter D., Loukas K., Julio L., Bruce L., and David O' H. 1999. Preliminary report on the design of a framework for distributed visualization. *PARALLEL AND DISTRIBUTED PROCESSING TECHNIQUES AND APPLICATIONS (PDPTA99)*. pp 1833-1839.

[8] Yang J., Shi J. Y., Jin Z. F., Zhang H. 2002. Design and implementation of a large-scale hybrid distributed graphics system. *4TH EUROGRAPHICS WORKSHOP ON PARALLEL GRAPHICS AND VISUALIZATION (2002)*. pp 39-49.

[9] David H. Porter, Paul R. Woodward, and Anusha Lyer. Initial Experiences with Grid-Based Volume Visualization of Fluid Flow Simulations on PC Clusters. University of Minnesota Supercomputing Institute Research Report UMSI 2004/114, July 2004 *http://www.lcse.umn.edu/hvr/VolVis-paper-4-28-04-for-our-Web-site.pdf*

[10] Stuart M. C., Nicolas S. H. and Malcolm M. 2003. Visualization in e-Demand: A Grid Service Architecture for Stereoscopic Visualization. PROCEEEDINGS OF THE UK ESCIENCE. SECOND ALL-HANDS MEETING, Nottingham Conference Center, *http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/161.pdf*

[11] Ken B., David D., Julian G., Musbah S., Jeremy W. and Jason W. 2004. Visualization in Grid Computing Environments. *PROCEEDINGS OF IEEE VISUALIZATION 2004*, pp 155-162.

[12] T.J. Jankun-Kelly, Oliver K., John S., Ma K. L., Bernd H., Kenneth I. J. and Bethel W. W. 2003. VisPortal: Deploying Web-based Visual Exploration Tools through a Web-Portal Interface. *PROCEEDINGS OF THE 3RD ANNUAL WORKSHOP ON ADVANCED COLLABORATIVE ENVIRONMENTS*.

http://www-vis.lbl.gov/Publications/2003/wace03.pdf

[13] T.J. Jankun-Kelly, Oliver K., John S., Ma K. L., Bernd H., Kenneth I. J. and Bethel W. W. 2003. Deploying Web-based Visual Exploration Tools on the Grid. *IEEE COMPUTER GRAPHICS AND APPLICATION*. Vol. 23(2): 40-50.

[14] Bilbao-Castro J. R., Marabini R., Carazo J. M., Garcia I. and Fernandez. J. J. 2004. The Potential of Grid computing in three-dimensional electron microscopy. *PARALLEL PROCESSING LETTERS*. Vol. 14(2): 151-162. World Scientific Publishing Company.

[15] Kreylos O., Tesdall A. M., Hamann B., Hunter J. K. and Joy K. I. 2002. Interactive visualization and steering of CFD simulations. ACM International Conference Proceeding Series. *PROCEEDINGS OF THE SYMPOSIUM ON DATA VISUALISATION 2002*. pp 25-34.

[16] Kong G., Stanton J., Newhouse S. and Darlington J. 2003.Collaborative Visualization over the Access Grid using the ICENI grid Middleware. *PROCEEDINGS OF UK E-SCIENCE ALL HANDS MEETING 2003*. pp. 393-396.

[17] J. Santon, S. Newhouse, and J. Darlington. 2002. Implementing a Scientific Visualization Capability within a Grid Enabled Component Framework. *EURO-PAR 2002,* LNCS 2400, pp. 885-888.

[18] Suzuki Y., Sai K., Matsumoto N. and Hazama O. 2003. Visualization Systems on the Information-Technology-Based Laboratory, *IEEE COMPUTER GRAPHICS AND APPLICATIONS*, Vol. 23(2): 32-39.

[19] Shigeru M., Eric B. L., Ma K. L., Ogata M. and Liu X. Z. 2003. A PC cluster system for simultaneous interactive volumetric modeling and visualization. *IEEE SYMPOSIUM ON PARALLEL AND LARGE-DATA VISUALIZATION AND GRAPHICS 2003*, pp 95-102.

[20] Render Nation. *http://www.rendernation.com/*

[21] Remote Render. *http://remote-render.com*

[22] Render Farm. *http://www.renderfarm.net/*

[23] Render Now. *http://rendernow.com*

[24] Render Core. *http://rendernow.com*

[25] ResPower. *http://respower.com*

[26] Render Rocket. *http://renderrocket.com*

[27] render IT. *http://render-it.co.uk*

[28] AXYZ Animation. *http://www.axyzfx.com*

[29] Sun N1 Grid Engine 6. *http://www.sun.com/software/Gridware/index.xml*

[30] Richard V. Dragon Case Study: Animation Grids. PC Magazine. Article - 2002-10-01 http://www.pcmag.com/article2/0,1759,530154,00.asp

[31] David P., Karissa M. and Mahmoud P. 2004. A holistic approach to high-performance computing: Xgrid experience. *PROCEEDINGS OF THE 32ND ANNUAL ACM SIGUCCS CONFERENCE. BALTIMORE, MD, USA*. pp. 119-124

[32] Apple Computer, XGrid. *http://www.apple.com/server/macosx/features/xgrid.html*

[33] Axceleon. *http://www.axceleon.com*

[34] United Devices. *http://www.ud.com*

[35] Uberware. *http://www.uberware.net*

[36] Discreet. *http://www.discreet.com*

[37] Images. *http://www.mentalimages.com*

[38] Tony A. 1995. Using RenderMan in animation production. SIGGRAPH 1995 Course 4, August 7, 1995. *www.renderman.org/RMR/Publications/sig95.course04.pdf*

[39] Tony A., Larry G., Matt P., Christophe H., Kevin B. and Lawrence T. 2001 Advanced RenderMan 3: Render Harder. SIGGRAPH 2001 Course 48. *www.renderman.org/RMR/Publications/sig01.course48.pdf.gz*

[40] Pixar. *http://www.pixar.com*

[41] Crockett, T. W. 1997. An introduction to parallel rendering. *PARALLEL COMPUTER*. Vol. 23(7): 819-843.

[42] Dieter K., Herbert R., Paul H. and Martin R. 2004. Interactive Virtual Reality on the Grid. *PROCEEDINGS OF THE 8TH IEEE INTERNATIONAL SYMPOSIUM ON DISTRIBUTED SIMULATION AND REAL-TIME APPLICATIONS (DS-RT '04)*, pp 152-158.

[43] Globus Toolkit. *http://www.globus.org/toolkit*

[44] Foster, I. and Kesselman, C. 1998. The Globus Project: A Status Report. *PROCEEDINGS OF HETEROGENEOUS COMPUTING WORKSHOP, IEEE PRESS, 1998*, pp. 4-18.

[45] Kalawsky, R. S. and Nee, S. P. 2004. Important issues concerning interactive user interfaces in grid based computational steering systems. *PROCEEDINGS OF THE UK E-SCIENCE ALL HANDS MEETING. 2004*. http://www.allhands.org.uk/2004/proceedings/papers/282.pdf.

[46] Xiang, H., Gong, B., Meng, X. X. and Kong, X. L. 2004. The design of adaptive platform for visual-intensive applications over the Grid. *PROCEEDINGS OF GRID AND COOPERATIVE COMPUTING 2003, LNCS*, pp. 172- 175.

[47] Rfc2616. http://rfc.net/rfc2616.html

[48] LELEWER, D. A. AND HIRSCHBERG, D. S. 1987. Data Compression. *ACM COMPUTING SURVEYS*, Vol. 19(3): 261-196.

[49] Huffman, D. A. 1952. A method for the construction of minimum-redundancy codes. *PROCEEDINGS OF THE INSTITUTE OF RADIO ENGINEERS*, 1098-1102.

[50] Khalid, S. 2000. *INTRODUCTION TO DATA COMPRESSION*, 2nd edition. Morgan Kauffman.

[51] Ziv J. and Lempel A. 1977. A universal algorithm for sequential data compression, *Proceedings of IEEE transactions on information theory*. Vol. 23(3):337-343.

[52] Salomon D. *Data Compression, 2000*. The complete reference 2nd Edition. Springer-Verlag.

[53] Ziv J. and Lempel A. 1977. A Universal Algorithm for Sequential Data Compression. *IEEE TRANSACTIONS ON INFORMATION THEORY*. Vol. 23(3): 337-343.

[54] Welch T. A. 1984. A technique for high-performance data compression, *IEEE COMPUTER*, Vol. 17(6):8-19.

[55] Bentley J. L., Sleator D. D., Tarjan R. E. and Wei V. K. 1986. A locally adaptive data compression scheme. *COMMUNICATIONS OF THE ACM*. Vol. 29(4):320–330.

[56] Yang E. H. and Kieffer J.C. 1996. Simple universal lossy data compression schemes derived from the Lempel-Ziv algorithm. *PROCEEDINGS OF IEEE TRANSACTIONS ON INFORMATION THEORY*. Vol. 42(1):239-245.

[57] Nasrabadi N. M., King R.A. 1988. Image coding using vector quantization: a review. *IEEE TRANSACTIONS ON COMMUNICATIONS*. Vol. 36(8):957-971.

[58] Wallace G.K. 1991. The JPEG still picture compression standard, *COMMUNICATIONS OF THE ACM. Vol. 34(4):30–44*.

[59] Taubman D. and Marcellin M. 2001. JPEG2000: Image Compression Fundamentals, Standards and Practice. *SPRINGER-VERLAG*.

[60] Zwicker E. and Fastl H. 1999. Psychoacoustics, Facts and Models. Springer Series in Information Sciences, 2nd edition.

[61] Stephenson I. 2002. Compressing RIB Sequences using diff, National Centre for Computer Animation, Bournemouth University, PIXAR User Group Meeting, San Antonio, SIGGRAPH 2002. *http://www.dctsystems.co.uk/Text/diffRibs.pdf*

APPENDIX B

FILE FORMATS

This appendix describes the structure of the .rib and .mi file formats and how the information is stored and the scanning process during the compression. A small section of the compressed log of the RIB and MI files are shown and explained.

RIB File Format

@bumble_cameraShape1.0.rib	*# First frame of animation – Identified by a @ sign. Our #decompression algorithm will create a new empty.mi file # named bumble _cameraShape1.0.rib and write all original #lines into this new file during the decompression stage.*

```
#Display Driver 0:
Display "bumble_cameraShape1.0.tif" "tiff" "rgb"

Format 768 576 1
Projection "perspective"  "fov" [37.8868 ]
Clipping 0.01 1000
Shutter 0 0
Transform [-0.774944 0.0923288 0.625249 0 1.04083e-016 0.989272 -0.146083 0 0.632029 0.113206 0.766631 0 -
0.18378 -1.94338 1.57849 1]
WorldBegin
TransformBegin
CoordinateSystem "worldspace"
TransformEnd
```

AttributeBegin
Attribute "identifier" "name" "|arches4|arches4Shape"
Transform [0.00139209 -0.0113279 -0.0159422 0 0.00565124 0.0155337 -0.0105441 0 0.0187226 -0.00384643 0.00436799 0 1.59859 0.641864 0.276381 1]
GeometricApproximation "motionfactor" 2
Color [1 1 1]
Opacity [1 1 1]
#Shaders
Surface "paintedplastic" "uniform string texturename" ["archtest.tx"] "uniform color specularcolor" [1 1 1] "uniform float roughness" [1] "uniform float Ks" [0.0862] "uniform float Kd" [0.5] "uniform float Ka" [1]
!772bd7e715a17f69189239142d68bc96PointsGeneralPolygons [……..……...……

……]

#During compression, lines containing
#'PointGeneralPolygons' in each .rib file will be
#scanned and hashed usingMD5 algorithm. A '!'
#character is appended to the 32 digit hexadecimal
#numbers hashed results. If the line is new, the 33
#characters string result will be appended to the
#beginning of the original 'PointGeneralPolygons' line
#otherwise it will only contain the hash result. The '!'
#indicator is used to identify the hashed line used by our
#decompression algorithm.

AttributeEnd
WorldEnd

@bumble_cameraShape1.1.rib

 # 2nd frame of animation – Identified by a @ sign. Our
#decompression algorithm will create a new empty.mi
#file named bumble _cameraShape1.1.rib and write all
#original lines into this new file during the
#decompression stage.

#Display Driver 0:
Display "bumble_cameraShape1.1.tif" "tiff" "rgb"

Format 768 576 1
Projection "perspective" "fov" [37.8868]
Clipping 0.01 1000
Shutter 1 1
Transform [-0.757529 0.0897328 0.646604 0 -9.71445e-017 0.990508 -0.137458 0 0.652801 0.104129 0.750339 0 -0.232002 -1.92936 1.55952 1]
WorldBegin
TransformBegin
CoordinateSystem "worldspace"
TransformEnd
AttributeBegin
Attribute "identifier" "name" "|arches5|arches5Shape"

Transform [-0.0189388 -0.00475999 0.00175422 0 -0.00494287 0.0188402 -0.00224166 0 -0.00114144 -0.00260757 -0.0193987 0 1.64685 0.605228 0.22073 1]
GeometricApproximation "motionfactor" 2
Color [1 1 1]
Opacity [1 1 1]
Surface "paintedplastic" "uniform string texturename" ["archtest.tx"] "uniform color specularcolor" [1 1 1] "uniform float roughness" [1] "uniform float Ks" [0.0862] "uniform float Kd" [0.5] "uniform float Ka" [1]
!772bd7e715a17f69189239142d68bc96 *# contains only the hash results. It contains the same #PointGeneralPolygons data found in #bumble_cameraShape1.0.rib file*

AttributeEnd
WorldEnd

@bumble_cameraShape1.2.rib *# 3rd frame of animation – Identified by a @ sign. Our #decompression algorithm will create a new empty.mi #file named bumble _cameraShape1.2.rib and write all #original lines into this new file during the #decompression stage.*

#Display Driver 0:
Display "bumble_cameraShape1.2.tif" "tiff" "rgb"

Format 768 576 1
Projection "perspective" "fov" [37.8868]
Clipping 0.01 1000
Shutter 2 2
Transform [-0.736982 0.0862501 0.670386 0 8.32667e-017 0.991825 -0.127606 0 0.675912 0.0940431 0.730958 0 -0.282936 -1.91257 1.54281 1]
WorldBegin
TransformBegin
CoordinateSystem "worldspace"
TransformEnd
AttributeBegin
Attribute "identifier" "name" "|arches4|arches4Shape"
Transform [0.00139209 -0.0113279 -0.0159422 0 0.00565124 0.0155337 -0.0105441 0 0.0187226 -0.00384643 0.00436799 0 1.59859 0.641864 0.276381 1]
GeometricApproximation "motionfactor" 2
Color [1 1 1]
Opacity [1 1 1]
Surface "paintedplastic" "uniform string texturename" ["archtest.tx"] "uniform color specularcolor" [1 1 1] "uniform float roughness" [1] "uniform float Ks" [0.0862] "uniform float Kd" [0.5] "uniform float Ka" [1]
!772bd7e715a17f69189239142d68bc96 *# contains only the hash results. It contains the same #'PointGeneralPolygons' data found in #bumble_cameraShape1.0.rib file*

AttributeEnd
WorldEnd

Mental Images (MI) file format

__ANTHONY_TOKEN__cars.1.mi *# First frame of animation – Identified by #__**ANTHONY_TOKEN**__. The decompression #algorithm will create a new empty.mi file named #cars.1.mi and write all original lines into this new #file during the decompression stage.*

Mayatomr 7.0.1.14m

min version "3.4"
link "mayabase.so"
link "mayahair.so"
$include "mayabase.mi"
$include "mayahair.mi"

#rendering parameters
…
…
…..
define 3D object parameter
object "log_home1|roof2|roofShape2"
visible on
shadow 3
transparency 3
reflection 3
refraction 3
finalgather 3
group
point vectors
[……
……
……]
normal vectors
[……
……
……]
tangent vectors
[……
……
……]
vertices, shared
[……
……
……]
polygons

[……
……
……] *#During compression, lines between 'group' and 'end group' in #each .mi file will be scanned and hashed usingMD5 algorithm. '<<<' #and '>>>' characters are appended to the start and ends of the 32 digit #hexadecimal numbers hashed results starting from the 2nd frame file #**__ANTHONY_TOKEN__cars.2.mi**. The '<<<' and '>>>' indicator #is used to identify the hashed line for our decompression algorithm. #During decompression, this data between 'group' and 'end group' will #be hashed and compared with the rest of the hashed strings in this #compressed.log file. If the hashed string is similar, the data will be #copied to the new #empty .mi file.*

end group
end object

__ANTHONY_TOKEN__cars.2.mi *# 2nd frame of animation – Identified by **#__ANTHONY_TOKEN__**. The decompression #algorithm will create a new empty.mi file named #cars.2.mi and write all original lines into this new #file during the decompression stage.*

Mayatomr 7.0.1.14m

min version "3.4"
link "mayabase.so"
link "mayahair.so"
$include "mayabase.mi"
$include "mayahair.mi"

#rendering parameters
….
…..
define 3D object parameter
object "log_home1|roof2|roofShape2"
visible on
shadow 3
transparency 3
reflection 3
refraction 3
finalgather 3
group
<<<***af0ba9f5f4712e6d9cefd8debaecac3e***>>> *# contains the hash results. It has the same #information found between 'group' and 'end # group' in **__ANTHONY_TOKEN__cars.2.mi** #file*

end group
end object

__ANTHONY_TOKEN__cars.3.mi *# 3rd frame of animation – Identified by **#__ANTHONY_TOKEN__**. The decompression #algorithm will create a new empty.mi file #named cars.3.mi and write all original lines #into this new file during the decompression #stage.*

min version "3.4"
link "mayabase.so"
link "mayahair.so"
$include "mayabase.mi"
$include "mayahair.mi"

#rendering parameters
…
…
…..
define 3D object parameter
object "log_home1|roof2|roofShape2"
visible on
shadow 3
transparency 3
reflection 3
refraction 3
finalgather 3
group
<<<af0ba9f5f4712e6d9cefd8debaecac3e>>> *#contains only the hash results. It has the #same information found between 'group' #and 'end group' in **#__ANTHONY_TOKEN__cars.2.mi file***

end group
end object

__ANTHONY_TOKEN__cars.700.mi *# Last frame of animation – Identified by **#__ANTHONY_TOKEN__**. The decompression #algorithm will create a new empty.mi file #named cars.700.mi and write all original lines #into this new file during the decompression #stage.*

min version "3.4"
link "mayabase.so"
link "mayahair.so"
$include "mayabase.mi"

$include "mayahair.mi"

#rendering parameters
…
…
…..
define 3D object parameter
object "log_home1|roof2|roofShape2"
visible on
shadow 3
transparency 3
reflection 3
refraction 3
finalgather 3

group
<<<***af0ba9f5f4712e6d9cefd8debaecac3e***>>> *# contains only the hash results. It has the #same information found between 'group' and #'end group' in #**__ANTHONY_TOKEN__cars.2.mi file***

end group
end object